CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Endangered Privacy: Identification of VPN-protected Video Streams At Scale

Master's thesis in Computer science and engineering

Martin Björklund

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Endangered Privacy: Identification of VPN-protected Video Streams At Scale

Martin Björkund

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Endangered Privacy: Identification of VPN-protected Video Streams At Scale
Martin Björklund

Supervisor: Romaric Duvignau, CSE
Examiner: Risat Pathan, CSE

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Endangered Privacy: Identification of VPN-protected Video Streams At Scale
Martin Björklund
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Despite the widespread adoption of HTTPS for enhanced web privacy, encrypted network traffic may still leave traces that can lead to privacy breaches. One such case concerns MPEG-DASH, one of the most popular protocols for video streaming, where video identification attacks have exploited the protocol's side-channel vulnerabilities. As shown by several works in recent years, despite HTTPS protection, the distinctive traffic patterns generated by DASH's adaptive bitrate streaming reveal streamed content despite encryption. However, these earlier studies have not shown that the vulnerability can still be exploited in large-scale attack scenarios and when the target is using privacy-preserving measures such as Virtual Private Networks (VPNs). To that end, this work presents a robust recognition system capable of identifying video streams even in the presence of a VPN or even when the attacker lacks network access; and demonstrates the attack over the largest dataset to date with a database exceeding 45,000 videos. By leveraging a combination of k-d tree search and time series methods, our protocol-agnostic framework surpasses all existing methods and achieves over 99.5% accuracy in real-time video stream recognition, even in noisy environments. We complement our work with an analysis of the vulnerability root cause when using adaptive bitrate streaming and propose a mitigation strategy to stand against such vulnerabilities. Since large-scale video identification can compromise user privacy and enable potential mass surveillance of video services, we release our tools and datasets to foster awareness and prompt timely solutions within the video streaming ecosystem to address these privacy concerns effectively.

Keywords: video privacy, DASH, privacy attacks, video stream identification, VPN

Acknowledgements

I would like to thank my supervisor, Romaric Duvignau, for his guidance throughout this project. My thanks also go to the staff of the Computer and Network Systems division for their supportive role in my research.

Martin Björklund, Gothenburg, 2024-06-13

Contents

List of Figures

List of Tables

1

Introduction

In the realm of user privacy preservation, HTTPS has become the standard, ensuring end-to-end encryption and enhancing privacy on the web. However, even encrypted network traffic leave traces that may be exploited to compromise users' privacy. In the last eight years, numerous video identification attacks (among others [5, 7, 9, 10, 20, 21, 23, 30]) have exploited a side-channel in the standardized MPEG-DASH protocol [25] (referred hereafter simply as DASH), revealing streamed content despite all traffic being encrypted. DASH, an adaptive bitrate streaming protocol, segments videos into variously encoded bitrate chunks. The purpose of segmentation is to optimize streaming, facilitating the use of existing HTTP infrastructure and allowing the adaptation of stream quality based on the client's network conditions. A significant drawback is that such segment-based streaming inadvertently generates a distinctive, bursty traffic pattern, which can be analyzed to deduce the content being watched by the client. Surprisingly, despite substantial privacy implications with a potential for mass surveillance of video services, not enough attention has been drawn to the issue. We believe the two key reasons behind such inattentiveness are (1) *non-scalable* recognition systems not capable to handle large datasets, and (2) *non-robust* recognition systems defeated by noisy packet captures e.g. due to network conditions or privacy-preserving measures. For example, as a privacy-preserving measure, streamers can mitigate such privacy intrusions by utilizing *Virtual Private Networks* (VPNs), which obscure the video service's IP addresses and blend the video traffic with other data within the same encrypted tunnel, defeating the strategy used by all known privacy attacks against DASH. However, as this thesis will highlight, these reasons are insufficient to overlook the problem. It deals with a concerning narrative, demonstrating that a scalable attack is possible even against such privacy-preserving measures, underscoring the immediate need for mitigative actions.

1.1 Related Work

Earlier video identification studies have assumed different attack models. In most studies, the attack has been employed at the network layer, where the attacker is assumed to have access to the transport layer information and encrypted payload carried between the streamer and the video service [7, 9, 14, 23, 29]. Such a scenario mimics that of an ISP or a network administrator in the role of the attacker. Other works have employed more covertly models working over LTE networks [5] or 802.11 [20] traffic. Aside from different attack models, known attacks fall into

Table 1.1: Summarized comparison of our work against previous demonstrated privacy attacks over encrypted video streams.

Year	Authors	Method[1]	Targeted platform(s)[2]	Without network access	Over VPN traffic	Dataset(s) with 10k+ videos	Capture starting anywhere in the video	Accuracy[3] >90% within 4min
2016	Reed *et al.* [20]	k-d tree	N	✓	✗	✗	✓	✗
2017	Schuster *et al.* [23]	CNN	A,N,Y,V	✗	✗	✗	✗	✓
2017	Dubin *et al.* [9]	NN, SVM	Y	✗	✗	✗	✗	✗
2017	Reed *et al.* [21]	k-d tree	N	✗	✗	✓	✓	✗
2019	Gu *et al.* [14]	DTW	Ah	✗	✗	✗	✓	✗
2020	Wu *et al.* [29]	L	F	✗	✗	✗	✗	✓
2020	Yang *et al.* [30]	Markov	Y	✗	✗	✗	✗	✓
2022	Bae *et al.* [5]	CNN	A,N,Y	✓	✗	✗	✗	✓
2022	Duvignau [10]	SVM	N,S	✗	✗	✓	✗	✗
2022	Khan *et al.* [4, 16]	CNN	Y	✓	✓	✗	✗	✗
2022	Wang *et al.* [28]	LCS	Ah	✗	✗	✗	✗	✗
2023	Björklund *et al.* [7]	k-d tree	S	✗	✗	✓	✓	✓
2023	Tang *et al.* [26]	BWD	Ah	✗	✗	✗	✗	✗
2023	Du *et al.* [8]	BOP	Y	✗	✗	✗	✗	✓
2024	Zhang *et al.* [31]	DTW	Y	✗	✗	✗	✗	✓
	This work	**k-d tree**	**A,H,S**	✓	✓	✓	✓	✓

[1]Employed search methods: Nearest Neighbor (NN), Support Vector Machine (SVM), Convolutional Neural Network (CNN), Dynamic Time Warping (DTW), Linear search (L), Markov chain (Markov), Bucket Word Dictionary (BWD), Bag-of-Patterns (BOP), Longest Common Subsequence (LCS).
[2]Targeted platform(s): Amazon Prime Video (A), Facebook (F), HBO Max (H), Netflix (N), SVT Play (S), Youtube (Y), Vimeo (V); Ad Hoc or unspecified streaming platform (Ah).
[3]If no capture duration is mentioned, we assumed the entire video is used as input to the recognition system.

two broad categories: Machine Learning (ML)-based methods and fingerprinting methods. Refer to Table 1.1 for a summary of known attacks in comparison to the recognition system introduced in this thesis.

ML-based studies [5, 9, 16, 23] have adhered to the traditional machine learning pipeline, i.e., streaming videos multiple times and extracting features from the resulting capture files to train a classifier, be it with deep learning [16, 23] or a combination of other models [9, 10]. The main advantage of using an ML approach is to let the system learn the characteristics of targeted videos by simply repeatedly watching them and do not require to have particular expertise in the targeted streaming protocol and video platform. ML works have been shown to be successful for modest datasets, but is bottlenecked by both data collection time (that can require watching the same video a hundred times under different network conditions [9, 30]) and cost of training models. Additionally, the ML-based attacker requires the videos to be streamed in the same timeframe and same quality seen in training. Hence all demonstrated attacks in this category assume, often implicitly, that the video is captured from its very start. Hence, all the aforementioned shortcomings present some major limitations of ML-based methods in scaling to large-scale attack scenarios. This thesis on the other hand presents an identification system that is capable of singling out a stream among more than 45k videos.

In contrast to ML, fingerprinting methods [14, 20, 21], have exploited that DASH exposes information about the sizes of its chunks to any client initiating playback. This leaked information can be used as ground truth reference data for videos which can then be exploited when comparing with live captured traffic. The advantage of such methods is that data collection becomes efficient since videos do not have to be streamed beforehand, but strategies must be developed both for aggregating encrypted packet sizes into larger chunks analogous to segment sizes, and searching

the already built fingerprint database. Different techniques such as k-d tree [7, 20, 21], dynamic time warping [14], and markov chains [30] have then been employed to find the correct video in a database based on fingerprints. Among all such searching techniques, the k-d tree has been shown to be the most scalable, with fast queries even for datasets of tens of thousands of identifiable videos. Using a searchable data-structure requires a distance measure, e.g. the Euclidean distance between a query point and a reference point stored in the data-structure, and is weakened by noisy inputs that can disproportionately affect the distance calculation. Those fingerprinting-based works assume the attacker captures traffic at the network layer where the conversation between the video service or its *Content Delivery Network* (CDN) and the streamer can be singled out from the user's other network conversations. With the video stream conversation singled out, these works estimate byte overhead from protocol headers and TLS encryption and remove it from the captured trace to approximate real segment sizes stored in the fingerprint database. This refinement makes the distance-based searches particularly effective. Additionally, some earlier works (e.g., [9, 21]) require TCP information in their methods, which are now invalidated by the recent shift to HTTP/3's QUIC protocol at the transport layer. Very few works [5, 16, 20] have attempted to remove these assumptions and any of them neither scale nor achieve high accuracy.

At last, let us observe that only a few studies [4, 16] have intended to extend recognition of videos beyond VPN-tunneling. Using a classifier based on a Convolutional Neural Network (CNN), their results over VPN traffic clearly show the challenge posed by VPN encryption with a drop from 99% accuracy (no VPN) to only up to 68% accuracy in recognition when considering a dataset containing the first 2 minutes from only 50-55 videos. The system demonstrated in this thesis does not suffer such a performance loss in the VPN scenario, and also scales effectively with a higher number of videos.

1.2 Contributions

In this thesis, we present the most robust and versatile system to date for video identification over encrypted packet traces, with demonstrated attack efficacy in real video stream captures over the largest published dataset of video fingerprints[1]. In detail, we combine the powerful logarithmic search of the k-d tree with a time series method to create a system capable of identifying videos at scale even in noisy environments. To show the capabilities of our system, we have collected a fingerprint dataset consisting of 45,000 unique videos and 550,000 fingerprints including the full libraries of two well-established video streaming platforms. Additionally, while previous research has highlighted DASH's leaky nature, this thesis demonstrates that the vulnerability is not exclusive to DASH. Our system is capable of also identifying videos streamed with HLS, DASH's main competitor and the primary adaptive bitrate streaming protocol used on Apple devices.

We say that the system presented in this thesis is *protocol-agnostic*; it relies on

[1] Publicly available link upon final submission.

the characteristics of adaptive bitrate streaming, not the network protocol used to deliver the stream. To demonstrate this property, we show that the attack holds in an authentic network environment where the target uses a VPN service (an active choice in privacy preservation). Additionally, we demonstrate that the attack is just as effective in a more covert scenario as a passive Wi-Fi (802.11) eavesdropper. In this scenario, the attacker is not on the network path but is within range to read encrypted Wi-Fi signals with a monitoring device, demonstrating an alternate attack vector. Our system can identify videos in real-time as they are streamed by the target and does not require the target and attacker to be in sync (i.e., the attacker can identify what the target is streaming no matter where in the video the target is). The only requirement is a continuous sniffing time of sufficient length. Additionally, and contrary to all previous works, the system has been built as a continuous identification tool and not a concluding algorithm, hence its opinion changes over time as the target switches video. Our identification system can identify and distinguish videos coming from several platforms and does not require to train a different classifier per platform.

Most of the code and tools developed for this project are made publicly available[2] as well as, upon request, 340 GB of capture files from our evaluation. Our aim by publishing our large-scale and efficient identification system is to raise awareness among all actors within major video streaming platforms that solutions to the privacy issue associated with adaptive bitrate streaming have to be considered and implemented immediately. Avoiding to patch the current video streaming ecosystem would certainly leave the door open to the implementation of large-scale surveillance. We explore mitigation solutions that alleviate the attack and we urge streaming developers to take them into account as early as possible.

1.3 Thesis Outline

In Chapter 2, we introduce the necessary background to our work, presenting ABR streaming (including the protocols DASH and HLS) and commonly employed privacy-preserving measures. Additionally, we introduce the targeted streaming services and the considered attack models. Chapter 3 presents our classification method in detail from data collection and building the fingerprint database to capturing traffic and identifying video streams. A thorough evaluation of our system is conducted in Chapter 4, followed by an exploration of mitigation strategies in Chapter 5. Chapter 6 discusses the limitations of the thesis and potential directions for future work. At last, Chapter 7 presents our conclusions.

[2]Open-source GitHub link provided upon final submission.

2

Background

This chapter provides essential background knowledge required to understand the system presented in Chapter 3. § 2.1 explores adaptive bitrate streaming, the dominant technique in modern video-on-demand streaming. § 2.2 delves into the mentioned leak and introduces the concept of fingerprints. Following this, § 2.3 discusses the privacy-preserving measures TLS, HTTPS, and VPN, which use cryptography to ensure confidentiality from eavesdroppers. § 2.4 examines the particular streaming services targeted in this thesis. Finally, § 2.5 takes a closer look at the VPN and Wi-Fi attack models, highlighting the challenges they present for the attacker.

2.1 Adaptive Bitrate Streaming

In the past, video and audio streaming was facilitated by protocols such as RTP and RTSP. These early protocols struggled with issues such as firewall traversal, lack of efficient caching mechanisms, and inability to adjust to varying network conditions due their fixed bitrate design. Recognizing these challenges, the industry has shifted towards *Adaptive Bitrate* (ABR) streaming protocols designed for performance in large distributed HTTP networks. ABR protocols work by segmenting the source media into smaller parts or chunks designated as *segments*, typically between two and fifteen seconds long, and encodes each segment at multiple bitrates. A client device will then request segments based on its available bandwidth and buffer consumption rate. At the start of a stream, the client downloads a *manifest file*, which contains different metadata, including the available encodings and the location of their corresponding segments. Since manifests and segments are static files, ABR streaming works seamlessly with CDNs where these files are cached at edge servers. Widely recognized and supported by Bitmovin's 7th Annual Video Developer Report [6], the industry leading ABR protocol implementations are *Dynamic Adaptive Streaming over HTTP* (DASH), also known as *MPEG-DASH*, and Apple's *HTTP Live Streaming* (HLS), with other options seeing minimal use.

2.1.1 DASH and HLS

DASH is the only ABR protocol that is an international standard [3]. It was developed by the Moving Pictures Expert Group (MPEG) with the goal of having an open standard alternative to HLS, which is proprietary to Apple. DASH is widely

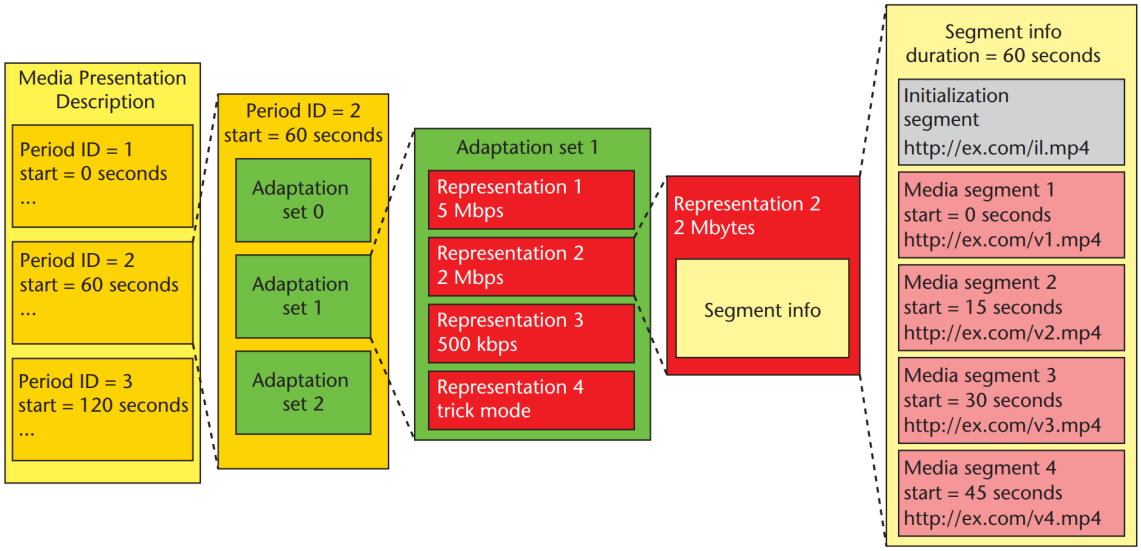


Figure 2.1: The hierarchical structure of a DASH manifest; source: Sodagar, 2011 [25].

supported by content providers and devices, with iOS devices being a notable exception. DASH's manifest file (MPD extension, for *Media Presentation Description*) is an XML-based document organized in a hierarchical structure (cf. Figure 2.1), and summarized hereafter:

1. **Media Presentation** (root): primary component holding all the other elements.
2. **Periods** (one or multiple): individual chapters of a video indicating strategic breakpoints where advertisements can be inserted.
3. **Adaptation Sets** (for each period): comprise various representations of the media (e.g. several encodings at various bitrate levels, all using the same video codec); different video codecs are stored in separate adaptation sets.
4. **Representations**: distinct versions of media within adaptation sets identified by an average bitrate or bandwidth and containing multiple segments; the representation is selected by the client based on its available bandwidth level.
5. **Segments**: references to the individual variable bitrate encoded segments that the client fetches and plays back; cause a distinguished network pattern during streaming, becoming the source of the leak (cf. § 2.2)

ABR streaming protocols define how the media segments are delivered from the server to the client, facilitated by the manifest. The segments themselves are wrapped in a container file format, with additional metadata, encryption if using Digital Rights Management (DRM), and a codec that indictates how the video data is compressed and decompressed. DASH's MPD typically references fragmented MP4 containers that are encryption and codec agnostic, supporting any encryption scheme and encoding format. While not directly related to the leak, this contrasts with HLS and is a reason for the emergence of the CMAF format, introduced in

the next section. The individual media segments can be part of a contiguous file, in which case the client sends HTTP byte-range requests to fetch them. We lend our terminology from the DASH Industry Forum [11] and call this mode *indexed addressing*. When indexed addressing is used, the MPD specifies the location of the file and the range of the *index segment* that contains the byte ranges of all other media segments. Segments can also be addressed with *explicit addressing* where segments are fetched via individual URLs that are either directly listed in the MPD or constructed using a template based on sequence numbers or explicitly defined time spans.

HLS [18] predates DASH and has wide support even though it is not an open standard. HLS manifests are represented by M3U8 files known as (master) *playlists*, that most often reference MPEG-TS (or simply TS) containers. While structurally different than an MPD, its purpose is nevertheless the same and the concepts of representations and addressing modes also exist in HLS.

2.1.2 CMAF

To be able to stream, devices and various software must support the streaming protocol and the underlying media and its encryption scheme. Consequently, to reach the widest possible audience, content providers package their content for both DASH and HLS with their respective common stacks, i.e. MP4 for DASH and TS for HLS. This inefficiency of both packaging and storing twice as much data is what the *Common Application Media Format* (CMAF) [1] was introduced to solve. CMAF allows for a single fragmented MP4 container based on the ISO base media file format to be referenced by both DASH and HLS manifests, making use of MPEG Common Encryption [2] that is agnostic to the DRM employed. Given that the referenced segments, as will be introduced next, are the source of the leak, it is implied that attacks on DASH can also become attacks on HLS if they reference the same segments (and use similar buffer management strategies; cf. Chapter 5).

In this thesis, we study well-established streaming platforms that employ DASH, HLS and CMAF. Furthermore, to keep consistency, we adhere to the DASH terminology but it is worth noting that the concepts also apply to HLS.

2.2 ABR Streaming Leak

When a client initiates a video stream, it will first request the manifest that facilitates the stream. During start-up, the client has an empty buffer and typically requests segments from a low bandwidth representation to ensure earliest possible playback. Once the client has a full buffer it reaches a steady state, where the time between segment requests depends on the buffer consumption rate. The client continuously estimates its bandwidth capacity to decide from which representation it should request segments from. If the client's bandwidth remains constant, it will keep requesting segments from the same representation, such as the best quality available if the client has a high bandwidth. This on and off period, due to requesting segments and downtime in between, causes a network trace with a pattern that

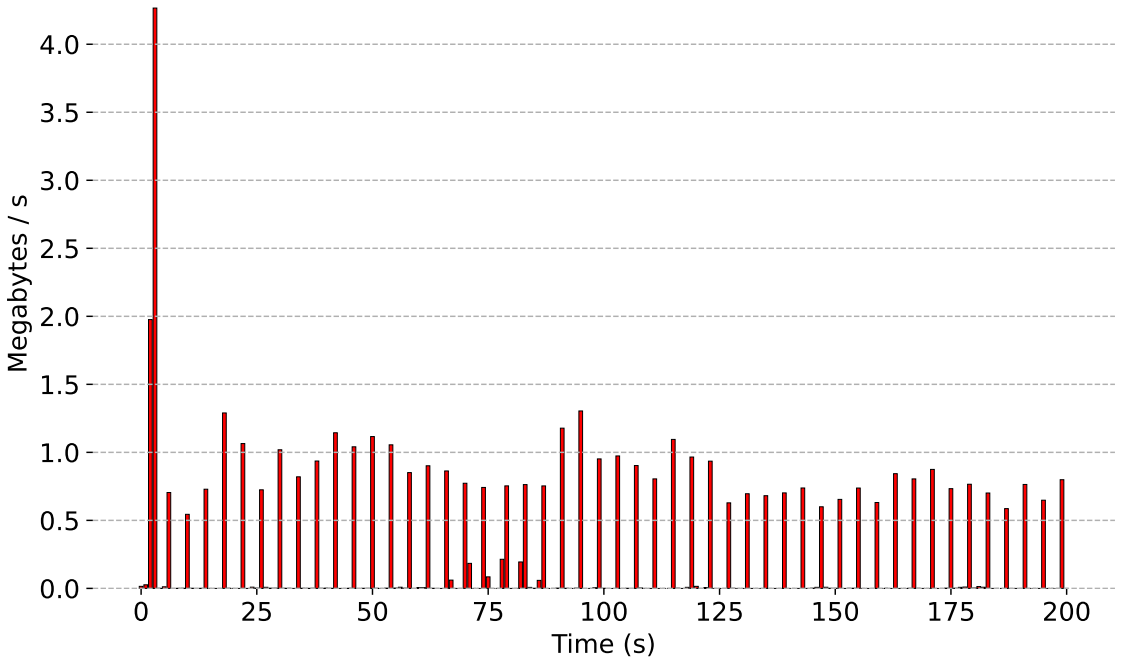


Figure 2.2: The *bursty* behavior of ABR streaming, starting off with the buffer-fill before transitioning to the steady state.

has been described as *bursty* [23], visualized in Figure 2.2. Each burst can be associated with the client requesting and downloading one or more segments. Because segments are encoded with a variable bitrate, their size depend on the content, with more bits allocated to complex segments (e.g. vibrant action scenes) and fewer bits to simpler segments (e.g. mostly static and dark scenes). The leak relies on the distinctive network trace that aligns with the order and sizes of the segments belonging to a specific video representation. Consequently, in order to perform the attack, previous works (and our work as well) assume that the target has a stable connection, allowing it to consistently request segments from the same representation.

2.2.1 Fingerprints

In previous works, fingerprints have had varied definitions. In machine learning works [5, 23], the observed network trace is labeled a fingerprint, and multiple streams of the same video is used as training data. In this thesis, we define a *fingerprint* as the complete sequence of exact segment sizes corresponding to a specific representation of a video. Consequently, a video available on a streaming platform will possess a unique fingerprint for each of its available representations (see Figure 2.3 for an example).

Our method, like previous fingerprinting methods [7, 14, 20, 21, 29], makes use of the ground truth fingerprint data to identify videos and does not require streaming videos beforehand because it is solely based on information derived from manifest files. To acquire a fingerprint, an attacker is required to first locate such a mani-

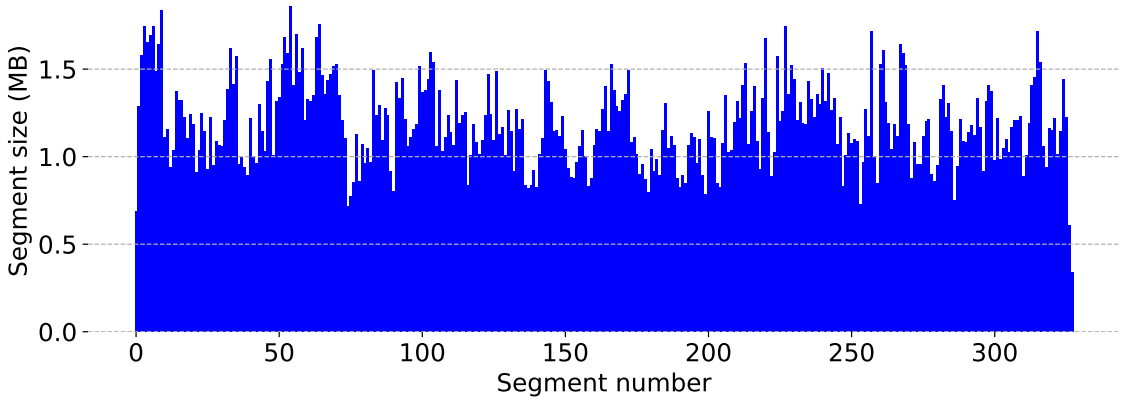


Figure 2.3: A fingerprint of *Friends S1E1* on HBO Max, belonging to a 720p representation with an average bandwidth of 3899124 bps.

fest file or an equivalent JSON schema (e.g. YouTube and Vimeo specific format equivalent to the MPD). In scenarios where indexed addressing is employed, the attacker must access the index segment to get the byte ranges that constitute the fingerprint while in explicit addressing, the information of each segment is retrieved via its corresponding URL. Once a fingerprint has been collected, it can be stored in a database and subsequently used as reference data by the attacker with captured network data, as will be explained in Chapter 3, which covers our method.

2.3 Privacy-preserving Measures

The foundation of the internet's communication relies on the Internet Protocol Suite, commonly referred to as TCP/IP. This suite, consisting of the foundational protocols TCP, UDP, and IP, was adopted in the 1970s and 1980s when the digital landscape was vastly different from today. The primary goal of these protocols were effective communication, with security considerations being secondary or even non-existent. The absence of security foresight in the original design has left these protocols open to various vulnerabilities. Despite this, the TCP/IP model remains the backbone of the internet we know today. To address the security gaps of the TCP/IP model, various supplementary protocols have been implemented on top, with one of the most prominent being TLS.

2.3.1 TLS and HTTPS

Transport Layer Security (TLS) is a cryptographic protocol standardized by the Internet Engineering Task Force (IETF) to ensure secure communication over a network. TLS, which builds upon the now-deprecated Secure Sockets Layer (SSL), has undergone several revisions, with the current version TLS 1.3 [22] defined in August 2018. TLS provides three important aspects of security; confidentiality, integrity, and authentication. Confidentiality ensures that data exchanged between parties cannot be read by unauthorized entities. This is achieved by encrypting messages

that only the other party can decrypt. Integrity guarantees that the data remains unaltered in transit so that it cannot be tampered with. For this, cryptographic hash functions are used to produce a unique fingerprint of the data to detect changes. Meanwhile, authentication is about establishing the identity of one or both parties. This is done using digital certificates to prove the validity of a public key belonging to a device, organization or individual.

Architecturally, TLS operates between the application and transport layers. This allows TLS to secure application data, but lower layer information such as IP addresses and TCP information remains unprotected. Although TLS can be implemented to secure a wide range of applications or application layer protocols, it is perhaps most recognized for securing the HTTP protocol, transforming it into Hypertext Transfer Protocol Secure (HTTPS), also known as HTTP over TLS.

Today, most web traffic is secured by TLS through HTTPS [13], and streaming platforms are no exception. Most previous works have assumed that the video streams are TLS-secured thereby benefiting from the confidentiality aspect it provides, but use the unprotected IP and TCP information to their advantage. Without that information (e.g. due to the use of a VPN), the attack becomes more challenging. However, as will be demonstrated in this thesis, our protocol-agnostic system remains effective even under such conditions.

2.3.2 VPN

Virtual private networks (VPNs) are commonly used to create secure connections between between computer networks, or a device and a network over insecure mediums, e.g. the Internet. Unlike HTTPS, which secures individual, unrelated sessions, VPNs encapsulate all incoming and outgoing traffic into a single, secure tunnel, ensuring consistent encryption and privacy across all applications. There are different VPN protocols available; for example, WireGuard is a modern protocol that aims to be more performant than traditional options like IPSec and OpenVPN.

There are several categories of VPN networks, but in this thesis we assume the role of a video streaming consumer and limit ourselves to commercial VPN services that advertise strong privacy guarantees and essentially provide proxy servers that use VPN technologies. The main benefit from a privacy-preserving point of view is that a network level observer situated between the client device and the proxy will see all device traffic go through a single encrypted tunnel. A stark difference between relying on TLS and the use of a VPN service, is that the latter is an active choice in preserving ones privacy, perhaps for a good reason. Consequently, this makes our method of identifying videos over VPN particularly concerning.

In our evaluation, we use NordVPN, a widely used VPN service solution that was ranked the best VPN service for privacy by `security.org` [27]. NordVPN uses a forked version of WireGuard called NordLynx, with some modifications to Network Address Translation (NAT) [15], but for our purposes, it is indistinguishable from WireGuard.

2.4 Studied Streaming Services

We evaluate our method on the entire libraries of two streaming services. One is *HBO Max*, an American subscription video streaming service. HBO Max is undergoing a merger with Discovery+ and the new service, Max, has launched in the United States and is currently being rolled out in Europe and Asia. The change in infrastructure means potentially malicious use of our published dataset is probably limited due to the videos being re-encoded. We also target *SVT Play*, the Swedish national public television broadcaster streaming service and the subject of a previous study [7] that includes a publicly available fingerprint dataset that we have reused and updated. Furthermore, HBO Max and SVT Play share the use of 4 second long segments, resulting in network traces with similar profiles. Consequently, it allowed us to explore the scalability of our method, as it would undoubtedly have been easier to distinguish another service with different segment durations.

2.4.1 HBO Max and SVT Play

HBO Max supports DASH, HLS and CMAF containers with indexed addressing. A video typically has 18 representations, but even with good bandwidth, device data is used to limit the representation quality, a common trait of streaming platforms that employ DRM. For example, even with high bandwidth exceeding the threshold for the best quality representation (e.g. 1080p or 4K), HBO Max videos played on a Linux web browser cannot receive a quality higher than 720p. HBO Max uses the most recent HTTP version HTTP/3 for compatible clients, which employs the more modern QUIC transport protocol in favor of TCP. In QUIC, much of the protocol state, including the information that would typically be visible in TCP with TLS (such as sequence numbers and acknowledgments), is encrypted. This negatively impacts the effectiveness of the attack methods of previous studies that use TCP information [9, 21]. We note that our system is agnostic to the transport mechanism and does not make such distinctions. Further discussion on the influence of the HTTP version is provided in Chapter 5.

SVT Play is a free service, and among the most popular streaming services in Sweden. It features news and current affairs on top of regular TV programs, shows and movies. Similarly to HBO Max, SVT Play also supports DASH, HLS and CMAF. However, SVT Play uses explicit addressing for segments, which requires a slightly different data collection approach that will be introduced in Chapter 3. An SVT Play video has 10 representations on average.

2.4.2 Other services

In contrast to ML methods that require training data, our system easily scales due to a straightforward data collection method. To showcase this feature, we also include the first season of *The Lord of the Rings: The Rings of Power* from Amazon's Prime Video, and demonstrate that our system can identify these videos inside the same large dataset. These representations were added manually by going to each episode and intercepting the manifest via browser developer tools. However, it would have

been possible to create an automated crawler to retrieve the entire library, as we have done for HBO Max and SVT Play. Our data collection method will be explained in detail in Chapter 3. Apart from HBO Max, SVT Play, and Prime Video, we have also identified a number of other platforms as directly vulnerable, further discussed in Chapter 5.

2.5 Attack Models

As previously mentioned, the attack model (*network attacker*) that is used in previous video identification published attacks most often assume that the attacker has full access to network-level information (e.g. TCP port/sequence/ack numbers and streaming service IP address in clear text, cf. Table 1.1). Large-scale attacks over a single platform [7, 21] have already been demonstrated for the network attacker model and is not further considered here. Instead, we assume in this thesis that the attacker is without such full network-level access, and demonstrate an attack on users who employ privacy-preserving measures such as a VPN, as well as an attack without network access, relying solely on physical proximity to the victim to read Wi-Fi signals. Remarkably, even with such weaker assumptions, our system does not suffer any performance loss compared to previous works, achieving 99.5% accuracy and precision on the largest dataset ever used for this purpose. Below, we explain both attack models in detail and outline additional assumptions common to both models.

2.5.1 VPN eavesdropper

In this model, the target streams while connected to a VPN service with the purpose of hiding both the video service and the video that is being watched. Here, the attacker sits on the network path between the target and the VPN proxy, for example a malicious ISP or network administrator. The attacker operates purely in observational capacity, rather than acting as a bridge, and reads the traffic at full throughput. Hence, only the VPN tunnel is visible to the attacker, which, as previously noted, encompasses all the internet traffic generated by the target, not just the video stream.

2.5.2 Wi-Fi eavesdropper

In this model, the target is streaming with HTTPS only (meaning the service would be identifiable by a network attacker) over a password protected Wi-Fi network that the attacker does not have access to. We assume that the attacker is nonetheless in physical range to read the Wi-Fi signals with a monitor device. Similar to the VPN scenario, the attacker will have to deal with traffic from several source networks on the same link, since they cannot distinguish between distinct IP conversations contained in each Wi-Fi frame's encrypted payload. This model demonstrates that even if the target has reason to believe that the network path is free of attackers, the stream can still be leaked through the air by just Wi-Fi signals. Additionally, it showcases our protocol-agnostic property, indicating that the attack is more about

data flow and the characteristics of ABR streaming rather than a particular medium or network protocol.

2.5.3 Common assumptions

For the case of both VPN and Wi-Fi eavesdropper, we assume that the target is not *actively* doing something else other than watching a single video from the streaming platform. This corresponds to the typical and realistic attack scenario we aim to reproduce, as most viewers are likely focused on the video itself, making additional online activities less probable during the streaming session. We also note that this assumption is enforced on some devices, e.g. smart TVs, due to only allowing one application to run in the foreground at a time. However, let us observe that the target's device is in a regular setting and generates regular amounts of background traffic. This is to mimic a real-world scenario, hence, we do not disable protocols or standard applications that would otherwise generate traffic. The attacker will have to deal with that traffic as additional noise, including data from the streaming services themselves that are not the actual stream, such as ad and telemetry data. We also assume that the target has a bandwidth of 100 Mbps and that a geographically close VPN server is chosen, which is the default for the selected VPN service that optimizes for speed. We further assume that the target maintains a stable connection, consistently requesting segments from the same representation during streaming, which is typical under such conditions. Note that in unstable network conditions, such as those found in some developing countries, frequent adaptation of stream quality can occur, resulting in switching representations often. Since our attack requires the target to request segments from the same representation for a certain period of time, we do not consider this scenario in this thesis.

3

Protocol-Agnostic Classification of Video Network Traces

Attack overview The attack consists of two phases. In the first, fingerprint data corresponding to the targeted platforms or videos are collected and preprocessed fragments of all fingerprints are stored in a k-d tree database. In the second or active phase, traffic from the target is captured and analyzed in real-time. Bursts present in the traffic are isolated and fed to an identification sub-system in charge of querying the fingerprint database. The identification logic assigns a score to candidates and when the highest score is beyond a predetermined threshold the system outputs a match. Contrary to previous works, the entire recognition system operates continuously and adapts its opinion over time. It is capable of identification after the target switches to another video and can accurately identify the video being streamed regardless of the playback starting point or when the system starts running.

Our method is explained in detail as follows. In § 3.1, we describe our data collection method. Next, in § 3.2, we explain how we organize and store the collected data into a k-d tree database of fingerprints. § 3.3 covers how we capture network data in real time and convert it into query points for our established fingerprint database. § 3.4 details how our system uses multiple queries as input to our identification heuristic to make the match. Finally, § 3.5 discusses the configurable system parameters and their default values.

3.1 Data Collection

The goal of data collection is to obtain fingerprints, i.e., segment sizes, from representations of videos. First, we need to obtain manifest files for each video, which contain information about its representations and how to access the segments. Since initiating a stream always requests the manifest first, previous fingerprinting works [21, 31] have typically used the WebDriver[1] interface for programmatic remote control of the browser to start playback and intercept and download the manifest of each video. However, this method is slow as it requires loading the entire page, executing JavaScript, and initiating the video (albeit still faster than ML that watch the entire video). We aimed to avoid the overhead of using web drivers. Therefore, we developed purely HTTP-based clients for both HBO Max and SVT Play by analyzing

[1] `https://www.w3.org/TR/webdriver2`

requests and replies for their respective internal APIs. These clients allow us to obtain metadata and manifests for all videos in seconds rather than hours, depending on the number of parallel connections we open and whether we run it from multiple hosts. We recall from § 2.1.1 that when using CMAF, DASH and HLS manifests can reference the same segments. Therefore, our clients are designed to retrieve only DASH manifests for simplicity and efficiency.

Once the manifest files have been retrieved, the method we use to obtain the fingerprints depends on the addressing mode used. For HBO Max, which uses indexed addressing, manifests will have a single URL for each representation and specify the byte range of the index segment. First, we send an HTTP range request[2] to retrieve the index segment. Then, once downloaded and parsed, the index segment contains the byte ranges of all other segments in the video, effectively giving us the complete fingerprint immediately. On the other hand, SVT Play uses explicit addressing. In this case, segments are not part of a contiguous file that supports random access; instead, they are independent resources fetchable via URLs, and the manifest provides information on how to construct these URLs. Since we cannot simply get a list of byte ranges to derive the fingerprint, we send HTTP HEAD requests to each segment URL and get its size from the `Content-Length` header field.

Our data collection program uses the outlined approach to automatically retrieve fingerprints from the full libraries of HBO Max and SVT Play. The only difference lies in the method of obtaining metadata and manifests, while obtaining fingerprints from manifests is done through a common interface. This means that more platforms can be added with relative ease. However, in this thesis, we limit ourselves to HBO Max and SVT Play due to time constraints, with a few Prime Video videos added manually to highlight generalization to other streaming services. Interestingly, while HBO Max and Prime Video are subscription services, manifests and segments are accessible unauthenticated from CDN edge servers, presumably because DRM decryption keys are required for playback. Since our goal is only to find out about representations and their segment sizes, we can gather most data entirely unauthenticated. This facilitates covert, straightforward data collection, even potentially using a distributed system to avoid IP-based throttling, and such activity would likely not seem out of the norm to these high-throughput servers. This method contrasts with machine learning efforts, where data acquisition often represents the most substantial bottleneck due to the requirement of actually playing the video.

3.1.1 Dataset

In total we collected a large dataset of 45k videos and 550k representations, the full libraries of HBO Max and SVT Play, see Table 3.1. 282 videos discarded on purpose because their very short duration (less than 1 minute). As mentioned we also retrieved the first season of *The Lord of the Rings: The Rings of Power* from Prime Video, consisting of 8 episodes with 11 representations each. The total data downloaded (not necessarily saved), including metadata, manifest files, index segments, and headers from HEAD requests, amounted to approximately 15 GB, which

[2] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests`

Table 3.1: Our dataset covering the full libraries of HBO Max and SVT Play, and eight videos for Prime Video.

Platform	# Videos	# Representations
HBO Max	17,769	322,133
SVT Play	27,269	227,263
Prime Video	8	88
Total	45,046	549,484

takes 20 minutes to download with a throughput of 100 Mbps. To limit potential malicious use of our toolkit, we keep our data collection program proprietary. However, we export our full dataset as a set of Tab-separated values (TSV) files (2.1 GB in total) that contain video metadata and the fingerprints for all videos on the targeted platforms.

3.2 Organizing the Data: *k*-d tree

Similar to some earlier works [7, 20, 21], we have chosen to store fingerprints in a k-d tree data structure. To achieve this, we preprocess our collected data to create a dataset $\mathcal{D}_k$ (fully defined below) suitable for input into the k-d tree. This preprocessing is outlined here.

Let $\mathcal{I}$ be the set of representations targeted by the attack; in our evaluation, $\mathcal{I} = \{1, 2, \ldots, 549484\}$. For each representation $i \in \mathcal{I}$, let $S_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,n_i})$ be its fingerprint, with $s_{i,j}$ representing the size of its j-th segment and n_i its total number of segments. Let the set of all segment sizes be denoted by $S = \{s_{i,j} \mid i \in \mathcal{I}, 1 \leq j \leq n_i\}$. We define the normalized fingerprint, with each element rescaled to be in the $[0, 1]$ range, as $S'_i = (s'_{i,1}, s'_{i,2}, \ldots, s'_{i,n_i})$ where $s'_{i,j} = \frac{s_{i,j} - \min(S)}{\max(S) - \min(S)}$. Furthermore, let $\partial S'_i$ be the discrete derivative of the normalized fingerprint S'_i, i.e., for $i \in \mathcal{I}$, $\partial S'_i = (\Delta s'_{i,2}, \Delta s'_{i,3}, \ldots, \Delta s'_{i,n_i})$, where $\Delta s'_{i,j} = s'_{i,j} - s'_{i,j-1}$ for $j = 2, 3, \ldots, n_i$. The reasoning behind computing the differences of each adjacent segment size is to make later comparison with imperfect data easier, explained further in § 3.4. For a given dimension $k \geq 1$ and representation $i \in \mathcal{I}$, we denote W_i^k the *window view* of width k over $\partial S'_i$, defined as:

$$W_i^k = \begin{bmatrix} \Delta s'_{i,2} & \Delta s'_{i,3} & \cdots & \Delta s'_{i,k+1} \\ \Delta s'_{i,3} & \Delta s'_{i,4} & \cdots & \Delta s'_{i,k+2} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta s'_{i,n_i-k+1} & \Delta s'_{i,n_i-k+2} & \cdots & \Delta s'_{i,n_i} \end{bmatrix}$$

The rows of the window view W_i^k can be conceptualized as each k-length sequence observed as a sliding window traverses $\partial S'_i$. We repeat the process for every normalized fingerprint S'_i to create the dataset $\mathcal{D}_k$ containing all window views $\mathcal{D}_k = \bigcup_{i \in \mathcal{I}} W_i^k$.

3.2.1 k-d tree

We store every row of every window view $W_i^k \in \mathcal{D}_k$ in a k-d tree $\mathcal{T}_k$, where the ℓ-th row $W_i^k[\ell] = \Delta s'_{i,\ell+1}, \Delta s'_{i,\ell+2}, \cdots, \Delta s'_{i,\ell+k}$ of W_i^k can be interpreted as a k-dimensional point in space. Our data-structure is populated with triplets of the form $\langle W_i^k[\ell], i, \ell \rangle$ with $W_i^k[\ell]$ being used to calculate distances between points as used in the k-d tree's $\mathcal{O}(\log n)$ lookup operation whereas (i, ℓ) is used to retrieve the video information corresponding to the k-dimensional point $W_i^k[\ell]$.

The `search` operation provided by our fingerprint database can be viewed as providing an implementation for a function

$$T_{\tau,r}^k : \mathbb{R}^k \to \mathcal{P}(\mathbb{N} \times \mathbb{N} \times \mathbb{R}),$$

that maps a k-dimensional point p to the τ closest points to p within radius r in $\mathcal{T}_k$. Technically, the k-d tree $\mathcal{T}_k$ is used to efficiently retrieve $\mathcal{T}_k(p, \tau)$ the τ closest points to p in the tree, before filtering out points outside of radius r. Formally, the output set $T_{\tau,r}^k(p)$ is made of triplets of the form $\langle i, \ell, d \rangle$ where $i \in \mathcal{I}$ is a video representation, $1 \leq \ell \leq n_i - k$ is a row number of W_i^k (i.e., $W_i^k[\ell]$ is a specific sequence of length $k+1$ within the video associated to the i-representation; 1 added due to the normalization step) and $0 \leq d \leq r$ is a distance such that:

$$\langle i, \ell, d \rangle \in T_{\tau,r}^k(p) \;\Leftrightarrow\; W_i^k[\ell] \in \mathcal{T}_k(p, \tau) \;\wedge\; d = \mathrm{dist}(p, W_i^k[\ell]) \leq r,$$

with $\mathrm{dist}(p, q) = \sqrt{\sum_{j=1}^k (p_j - q_j)^2}$ being the euclidean distance between the two k-d points $p = (p_1, \ldots, p_k)$ and $q = (q_1, \ldots, q_k)$. The introduced output format is relevant for the identification heuristic outlined further in § 3.4.

3.3 Capturing Traffic: Packets and Bursts

The objective of the capture is to aggregate packet sizes into chunks corresponding to the actual segment sizes downloaded during video streaming. These chunks can subsequently be used to query the k-d tree to obtain a shortlist of potential candidates (all within a certain radius). As we recall from § 2.5, the network attacker model assumed by many previous works allows the conversation between the target and the streaming service to be singled out and analyzed. Transport layer information, header sizes and TLS-overhead can be used to precisely estimate the real segment sizes. In this work, we assume less potent attacker models where this information is unavailable. Below, we outline how our dedicated tool converts traffic consisting of network packets or Wi-Fi frames into query points for our fingerprint database.

3.3.1 VPN eavesdropper scenario

In this scenario, we assume the VPN eavesdropper attack model (cf. § 2.5.1), and we monitor all network packets passing us by. As previously mentioned, while the payload of each packet is potentially encrypted and unintelligible, information such as source and destination addresses, as well as port numbers for TCP and UDP-based traffic, remains readable and is required to route the packet to its intended

Table 3.2: 10-second snapshot of BurstShark output as target 10.0.1.42 watches an episode of *The Last of Us* on HBO Max.

Timestamp	Source	Destination	# Bytes
1701877603.265	10.0.1.42:50868	79.142.77.112:51820	11272
1701877603.878	79.142.77.112:51820	10.0.1.42:50868	2503928
1701877607.235	10.0.1.42:50868	79.142.77.112:51820	10608
1701877607.867	79.142.77.112:51820	10.0.1.42:50868	2210188
1701877611.711	10.0.1.42:50868	79.142.77.112:51820	6232
1701877611.774	79.142.77.112:51820	10.0.1.42:50868	909000

destination. VPN traffic adds the challenge that packets from different sources (e.g., the video stream versus another server) will have the same structure, sharing the same source, destination, and ports. Our solution consists of binning conversations into the classic 5-tuple source IP address/port number, destination IP address/port number and the protocol in use. For example, WireGuard does not provide obfuscation by default and typically runs on port 51820, but we do not make such distinctions as our system should work on any protocol, regardless if a VPN is used or not. On each network flow we aggregate sequences of packet sizes (which are discernible even if encrypted) into chunks based on throughput. This is done using a dedicated tool we have developed called BurstShark[3].

BurstShark wraps TShark, the terminal version of the widely used packet analyzer software WireShark. It reads packets, makes sure they carry application data, and does this aggregation in real-time, without the need for post-processing. The output of BurstShark is similar to TShark, but instead of packets it outputs aggregated packet sizes over short intervals that we call *bursts*; see Table 3.2 for an example of output. This allows us to carry out the attack and identify videos in real-time by querying our fingerprint database as it creates bursts. While BurstShark can perform some data cleaning, such as discarding bursts that could not possibly be associated with a video segment due to size, and accounting for the header sizes on the outermost layer, it is far from rigorous. For VPN flows, we cannot know if packets unassociated with the video stream are interleaved in a burst. Accepting that the data will contain such noise, we rely on our identification heuristic to manage it. This approach is one of our system's salient features, enabling it to perform effectively even with noisy data; unlike previous work that often emphasizes rigorous data cleaning at the network level.

3.3.2 Wi-Fi eavesdropper scenario

Here, we assume the Wi-Fi eavesdropper model (cf. § 2.5.2), and use a monitor device to read Wi-Fi (802.11) signals. In this case, BurstShark bins conversations into a tuple consisting of the source MAC address (e.g. wireless access point) and target MAC address (e.g. streaming laptop). Capturing 802.11 data frames is not always trivial; while control and management frames can be simpler to catch, data

[3] `https://github.com/embeage/burstshark`

frames will usually be transferred at full speed. The monitor mode device must match the capabilities of the transmitting device such as frequency, bandwidth and spatial streams. Based on the equipment we have available, we limit ourselves to 2.4 GHz 802.11n set to a fixed channel. Despite this, it is common to miss sequences of frames. However, since we only need to know the volume of data transmitted, we estimate the missing bytes by using the sequence numbers, noting which ones are missing and assuming each missing frame contains the maximum transmission unit (typically 1500 bytes). Other than accounting for missed frames due to the more unreliable medium, BurstShark produces bursts for each Wi-Fi conversation in the same manner as in the network path (and VPN eavesdropper) scenario, aggregating the sizes of frames (or packets) into appropriate query points for the fingerprint database. This process will be explained in detail in the upcoming section.

3.4 Identification Logic

This section details how bursts produced by BurstShark are converted into the system's predictions; which video is being streamed at each timestep. It introduces a number of system parameters, whose values and trade-offs will be discussed in the next chapter on system configuration. This includes the parameters k, r, and τ related to the k-d tree introduced in § 3.2.

3.4.1 Querying the k-d tree

Let $X = (x_t)_{t\geq 1}$ be an unbounded series of bursts produced by BurstShark at discrete time steps; timestep t is referred as "time t" whenever the context makes it unambiguous. As for ground-truth datapoints (see the preprocessing steps outlined in § 3.2), let $X' = (\Delta x'_t)_{t\geq 2}$ be the discrete derivative of the normalized burst sequence X, i.e. $\Delta x'_t = x'_t - x'_{t-1}$ for $t \geq 2$, where $x'_t = (x_t - \min(S))/(\max(S) - \min(S))$, where S is the set of all segment sizes in our dataset as defined in § 3.2.

The reason behind manipulating the *differences* instead of the absolute numbers is to eliminate potential constant overhead present in adjacent bursts. For example, while the video is variable bitrate encoded, audio is typically constant bitrate and has a similar size in each burst. Also, the added noise due to background network traffic will mostly on average cancel out if we assume the volume of background traffic stays roughly constant over time (at least when aggregated over segments duration). We define the *query k-d point* X^k_t at time step $t \geq k+1$ as follows: $X^k_t = [\Delta x'_{t-k}, \Delta x'_{t-k+1}, \ldots, \Delta x'_t]$ is the derivative of the normalized k+1 last recorded bursts at time $t \geq k+1$. Note X^k_{t+1} is efficiently calculated from X^k_t using a sliding window of length $k+1$ over the bursts sequence.

At last, we query our fingerprint database (that was already built for a specific value of k before BurstShark started) to retrieve the set of triplets $T^k_{\tau,r}(X^k_t)$ following the fixed parameters τ and r used to configure the system; see § 3.2 for the definition of $T^k_{\tau,r}$. We call $C^k_t = T^k_{\tau,r}(X^k_t)$ the set of *candidate triplets* at time step t.

3.4.2 Grouping candidate triplets into alignment groups

In previous k-d tree works [7, 20, 21], a single query was sufficient to confidently declare an identification, with each query treated independently without regard to queries at different points in time. Due to isolation of the conversation at the transport layer and data cleaning, a query point could be almost identical to its counterpart stored in the tree. In this work, we cannot rely solely on a single query point due to the increased noise, which generates too many false positives (see our comparison with the previous method in Chapter 4). Instead, we introduce the concept of *candidates* (distinct from candidate triplets), which will be explained in the remainder of this section. To provide an intuitive sense of candidates, an illustrative example is included at the end of the section.

In detail, we group candidate triplets $\langle i, \ell, d\rangle \in C_t^k$ by their representation i and *alignment* $j = t - \ell$ into the *alignment group* $(i, j) \in \mathcal{I} \times \mathbb{Z}$. We say that two candidate triplets $\langle i_a, \ell_a, d_a\rangle \in C_{t_a}^k$ seen at time step t_a, and $\langle i_b, \ell_b, d_b\rangle \in C_{t_a}^k$ seen at time step $t_b \geq t_a$ are *aligned* if

$$i_a = i_b \quad \wedge \quad \ell_b - \ell_a = t_b - t_a.$$

According to the *time series lag* parameter L, an alignment group $A(i, j)_t$ "at timestep t" is made of all candidate triplets seen during the last L time steps that belong to the alignment group (i, j) – all such triples are then aligned with each other (note that being aligned is indeed transitive), i.e.,

$$A(i, j)_t = \bigcup_{t-L \leq t' \leq t} \{\langle i, \ell, d\rangle \in C_{t'}^k \mid t' - \ell = j\}.$$

We call *candidate* at time t any non-empty alignment group $A(i, j)_t$ at time t, with $\mathcal{C}_t = \{A(i, j)_t \mid A(i, j)_t \neq \emptyset\}_{i \in \mathcal{I}, j \in \mathbb{Z}}$ being the set of candidates at time t. An alignment group (i, j) is said to be *present* at time t if a triplet corresponding to that alignment group has been observed for time step t, i.e., $\langle i, t - \ell, d\rangle \in A(i, t - \ell)_t$ for some $d \in \mathbb{R}$. In other words, the set of candidates $\mathcal{C}_t$ at time t correspond to all alignment groups that have been present at least once in the most recent L timesteps.

Example: To give an example of candidates, assume the target is watching some video (representation) at timestamp 1:15:30 and one of the triplets obtained by querying our k-d tree identifies this part of the video. Now, assume there are no matches, i.e. candidate triplets, for the video in the next minute because of noise. Then, one minute later a match for the same representation at timestamp 1:16:30 is retrieved, correlating with where the target would be in video, assuming uninterrupted playback. Our main underlying hypothesis is that this does not appear out of mere luck. Hence, whenever candidate triplets are aligned, it is a very good indication that the considered representation likely belongs to the video being played. Note we use real timestamps for the sake of the example as opposed to the discrete time steps of our burst-based system.

3.4.3 Scoring

Recall, we have so far built at time t a set of candidates $\mathcal{C}_t$ from which we need to decide if one of the candidates c produces a *match*, i.e., if our recognition system is confident enough in the candidate c to output it for time step t. To decide for a match, we rank all candidates of $\mathcal{C}_t$ using a heuristic scoring function. Intuitively, the score of a candidate indicates the likelihood of the candidate to correspond to the actual video being played with the following aspects increasing the score of a candidate: repetitive presence of the candidate in previous timesteps and closeness of previous query points to the candidate when it is present.

Let us now explain our scoring function $\texttt{score}(c,t)$ for any candidate $(i,j) \in \mathcal{C}_t$ in detail. For each candidate $c = (i,j) \in \mathcal{C}_t$ that is present at time step $t \geq k+1$, i.e., such that $\langle i, \ell, \text{dist}(X_t^k, W_i^k[\ell])\rangle \in A(i,j)_t$ with $j = t - \ell$, we assign first a base score D_t^c with $0 < D_t^c \leq 1$ and determined by the exponential distance decay from the query point X_t^k at time t, i.e., calculated as $D_t^c = e^{-\lambda_{dist} \cdot \text{dist}(X_t^k, W_i^k[\ell])}$, where the constant λ_{dist} represents the *distance decay factor*. If a candidate $(i,j) \in \mathcal{C}_t$ is not present at time t then we set its base score to zero, i.e. for any alignment group $c = (i,j)$ we have $\neg(\exists d \in \mathbb{R}, \langle i,j,d\rangle \in C_t^k) \implies D_t^c = 0$. The base scores vector $\mathbf{D_t^c}$ for candidate $c \in \mathcal{C}_t$ at time step t is defined as $\mathbf{D_t^c} = [D^c_{\max\{k+1,t-L+1\}}, \ldots, D^c_{t-1}, D^c_t]$ and contains the L most recent base scores for c; note if $t < L$, the vector $\mathbf{D_t^c}$ is of smaller size. Furthermore, we introduce a vector $\mathbf{W} = [w_1, w_2, \ldots, w_L]$ made of L weights where $w_1, \ldots, w_{L-k-1}$ are all set to a fixed constant $w \geq 1$ and the remaining $k+1$ elements $w_{L-k}, \ldots, w_L$ are set to 1. The main input for the score for any candidate c present at time t is the weighted sum $S_t^c = \mathbf{W} \cdot \mathbf{D_t^c}$ of the base scores by the fixed weights. As our system is non-conclusive it needs to adapt its opinion over time, therefore we also log scale S_t^c and apply the *decay constant* λ_{score} to a candidate's previous score when it is not present in t. The score of candidate $c \in \mathcal{C}_t$ at timestep t is thus calculated as:

$$\texttt{score}(c,t) = \begin{cases} \ln\left(1 + \sum\limits_{\substack{n\geq 1,\\ n \geq k+1+L-t}}^{L} D^c_{t-L+n} \cdot w_n\right) & \text{if c is present at time t} \\ \texttt{score}(c,t-1) \cdot \lambda_{score} & \text{otherwise} \end{cases}$$

The behavior of the scoring function over time is illustrated in Figure 4.2 in the evaluation chapter. This figure demonstrates how the score increases with reappearing candidates and drops off during both noisy periods and when the streamed video is changed.

Summary and justifications: In brief, the score of alignment groups that are absent of the last L timesteps is 0. Among candidates, when not present, the score of a candidate decreases exponentially by a factor λ_{score} at every time step. If present, the score is the product of the base scores of the time steps when it appears in the last L steps, weighted so that the most recent $k+1$ scores have a lesser importance. The prioritization of older base scores over recent ones is illustrated in Figure 3.1. We motivate this decision by assuming a *false* candidate c has been produced by the query point $X_t^k = [\Delta x'_{t-k}, \Delta x'_{t-k+1}, \ldots, \Delta x'_t]$. As the sequence evolves

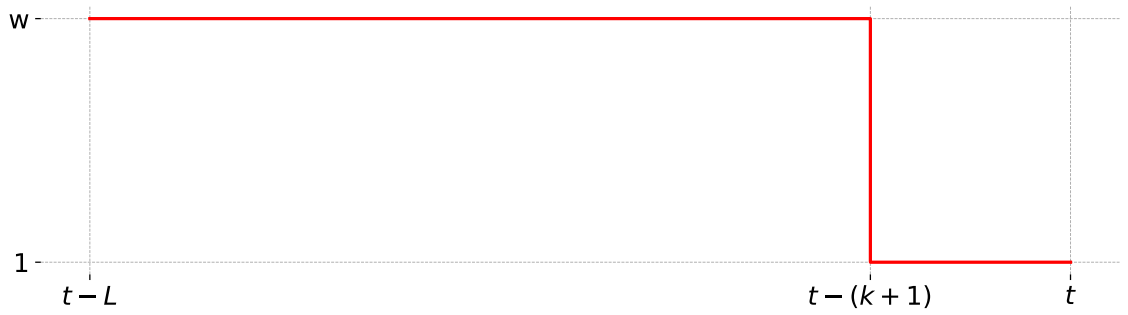


Figure 3.1: Illustration of weights for some w, k and L emphasizing recent scores as less important.

to $X^k_{t+1} = [\Delta x'_{t-k+1}, \Delta x'_{t-k+2}, \ldots, \Delta x'_{t+1}]$, all elements in X^k_t exist in X^k_{t+1} except for the discarded element $\Delta x'_{t-k}$ and the added element $\Delta x'_{t+1}$, highlighting this latter one as the focal point for determining the likelihood of c reappearing as a result of X^k_{t+1}. With each new time step, the influence of the initially overlapping elements on the likelihood of c diminishes. This diminishing likelihood can be expressed as $\mathbb{P}(c \mid X^k_{t+1}) > \mathbb{P}(c \mid X^k_{t+2}) > \ldots > \mathbb{P}(c \mid X^k_{t+k})$. Consequently, we place less emphasis on recent base scores, and our empirical data have shown this to be effective in preventing high scores for false candidates.

3.4.4 Prediction

In each time step t we select the highest score candidate above a predefined threshold θ which becomes the prediction. If no candidate has a score above the threshold, then the system does not make a prediction, instead returning -1 to signal that it is not confident enough. Additionally, let us observe that because a candidate also includes the alignment $j = t - \ell$, we can derive the playback position of the target by using ℓ, the segment duration and accounting for the buffer length used by the platform.

3.5 System Configuration

Table 3.3: Configurable system parameters and default values.

Variable (§ defined)		**Value**	**Description**
k	(§ 3.2)	8	Dimension of the k-d tree
r	(§ 3.2)	0.025	Radius of the k-d tree search
τ	(§ 3.2)	50	Max elements retrieved from $\mathcal{T}_k$
L	(§ 3.4)	75	Time series lag
w	(§ 3.4)	3.0	Weight for $t - L, \ldots, t - (k + 1)$ scores
λ_{dist}	(§ 3.4)	100.0	Decay constant for (distance) base scores
λ_{score}	(§ 3.4)	0.9	Decay constant for candidate scores
θ	(§ 3.4)	1.9	Confidence threshold

3.5.1 System parameters

Earlier k-d tree works [7, 20, 21] have used a value for k between 3 and 6, noting a trade-off between identification time and accuracy. It is known that k-d trees suffer in performance as k grows, a concept known as the *curse of dimensionality*. In this work, we did not have query points of sufficient quality to obtain accurate queries with a $k < 6$. We found that an 8-dimensional tree was more accurate than 6 and 7, and that identification time differences were negligible due to our method requiring more than a single point to make a prediction. The accuracy of the search was met with diminishing returns at $k > 8$, with cost in both query time and memory use. We note that our heuristic works by assuming points belonging to the representation we are searching for will, over time, appear more often and with a closer distance than other points. The previous k-d tree works only needed a single query point to surpass a set threshold based on the Pearson correlation with one of the returned points in order to accurately declare a match. In our method, the overlap between true and false points greatly reduced the effectiveness of the previous strategy, as demonstrated by our comparison presented in Chapter 4.

We use a radius value r of 0.025 which can be thought of as a generous upper-bound, with most lookups being limited to the τ closest points, which we keep at 50. r and τ complement each other in that r limits the lookup in sparse regions of the tree, while τ limits more dense regions. For example, as noted in earlier k-d tree works [7, 21] some videos have periods of darkness, resulting in a sequence of equal sized segments. Our approach includes computing the differences between adjacent segment sizes. The derivation of such sequences consequently produce 9,212 identical points of the form $\langle 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. However, these belong to only 31 videos, and majority of points correspond to darkness or static imagery at the end of videos. Furthermore, our implementation is not as sensitive to such sequences due to requiring multiple points and prioritizing older scores, for which we use $w = 3.0$. This lessens the impact of false candidates appearing in clusters, as explained in § 3.4, and consequently improves accuracy[4], identification time and the system's ability to retain its predictions, by allowing a lower threshold to be set. We use a lag of $L = 75$, ensuring that the system retains only the most recent 75 time steps without indefinitely consuming memory, and further increasing it had no significant impact on accuracy.

For the base scores, we use $\lambda_{dist} = 100.0$, resulting in base scores close to 0 for points closer to r. Note the difference with tightening the r-value, since in that case the candidate would not be considered as present, while λ_{dist} allows us to both give low scores to false candidates, while simultaneously accounting for previous candidates appearing again but with a bit more noise. We set $\lambda_{score} = 0.9$, setting a lower value technically allows the system to adapt faster to videos changing, but consider that a sequence of $k + 1$, i.e 9 bursts are required at minimum to initially predict a new video. With the suggested parameters, even when the system is the most confident, the score will converge around 5.0, allowing it to drop to approximately 1.9 during the transition period. As previously noted, this can be visualized in Figure 4.2

[4] About ~3% in our parameter tuning evaluation, not shown in this work.

presenting score over time, where the threshold θ is also set to 1.9. Uncertainty due to noise is characterized by the score dropping, before the system sees the candidate present again. The effect of different thresholds is evaluated in Chapter 4.

3.5.2 Implementation

Our implementation of the identification logic uses common Python libraries such as NumPy, pandas, and the k-d tree of scikit-learn[5]. The 8-dimensional tree contains 239,553,654 points derived from the 549,484 fingerprints and uses approximately 19.3 GB of memory. A lookup, i.e. call to the function $T^k_{\tau,r}$, takes about 13 ms on average on our midrange test system, equipped with an Intel Core i5-10600K CPU @ 4.10 GHz and 64 GB of DDR4 RAM, running Ubuntu 22.04 LTS. Being in the order of milliseconds, for our evaluation it is deemed negligible against the playback (wall) time. Our exported TSV dataset can be loaded by the program and used in conjunction with our capture data to reproduce the experiments presented in Chapter 4.

[5]Open-source GitHub link provided upon final submission.

4

Evaluation

We present in this chapter a thorough evaluation of the recognition method presented in Chapter 3 and parameterized by the default values as presented in Table 3.3 and explained in § 3.5. § 4.1 first details the real network environment that is used for our evaluation, followed by the presentation of our results in § 4.2.

4.1 Experimental Setup

4.1.1 Network environment

Our aim is to mimic a typical real life streaming scenario, therefore we setup a regular home network, see Figure 4.1, with an internet-facing router. LAN devices, i.e. the *targets*, connect to the network wirelessly through a separate access point. Between the router and the access point we setup a layer 3 switch with monitoring capabilities. We connect the *SPAN port eavesdropper* (analogous to the VPN eavesdropper model cf. § 2.5.1) to the switch, such that it receives a copy of all packets on the Access Point (AP) port. While the eavesdropper is part of the LAN, it could be deployed anywhere on the network path between the target and the CDN server providing the stream, for example at the ISP. In the Wi-Fi attack model (cf. § 2.5.2), we employ the *Wi-Fi eavesdropper* device with a wireless network card in monitor mode, without access to the LAN. In both attack models, the targets are unaware that there is an eavesdropper on the LAN.

For the VPN evaluation, there are two target laptops streaming simultaneously, a MacBook Pro running macOS 11 and an Acer Swift running Windows 11. This

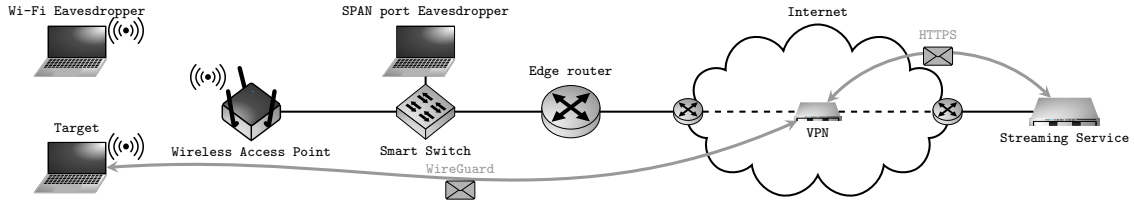


Figure 4.1: Illustration of the attack and testing environment; note that the testing setup has been designed to facilitate our evaluation and most elements are not required for the attack to be performed.

Table 4.1: Streams played per platform and device.

(a) VPN streams captured by SPAN port eavesdropper.

Platform	MacBook	Acer	Total
HBO Max	59	233	292
SVT Play	150	217	367
Prime Video	8	0	8
Total	217	450	667

(b) Non-VPN streams captured by Wi-Fi eavesdropper.

Platform	MacBook	TV	Total
HBO Max	118	3	121
SVT Play	149	0	149
Total	267	3	270

setup demonstrates the applicability of our method across different hardware and operating systems (along with being less time-consuming). Both devices are fully updated and use the default OS-level settings. As mentioned, we purposefully try to emulate a typical user, so the MacBook is signed in with an Apple ID, and the Acer is signed in with a Microsoft account. Consequently, each device generates some idle network traffic, such as syncing, telemetry, and checking for updates. Each device is connected to a VPN proxy using the native NordVPN application for macOS and Windows respectively, and we use a kill switch to ensure internet access is blocked if the VPN connection drops. A shared web driver program for the Chrome browser is used to navigate each platform and start playback automatically. In addition, three HBO Max HLS streams using Safari and eight Prime Video streams were streamed manually. In the Wi-Fi evaluation, we keep using the MacBook, now without VPN, and we manually play another three streams on a 2016 Samsung TV to further demonstrate that the system generalizes to other devices.

4.1.2 Testing set

Along with the manually played videos, we have selected random[1] videos from HBO Max and SVT Play for each device to stream. Each video is streamed for 10 minutes, starting from a random starting position within the video so that there is at least 10 minutes of remaining content to play. Additionally, because our system is designed to be continuous and adaptive to changes, most streams were done in sessions of three videos. In each evaluation setting, the eavesdropper runs our capture program for the full session, but for reference the original capture (PCAP) files are also kept.

Importantly, the system is capable of real-time identification but the evaluation is done by saving the capture files since we want to try different configurations on them as well as keeping them available to the public. Note, however, that there is no real difference in the algorithm, and evaluation results for a certain configuration apply to the real-time use-case as well. Table 4.1 shows the number of streams per platform and device in both evaluation settings. Discrepancies in the distribution relate to the script often failing to initiate playback. For example, on HBO Max

[1]Selecting a video at random is not a feature of any of the considered platforms but is made straightforward using the video (fingerprint) database that we have collected. In our study, we selected random videos of at least 20 minutes.

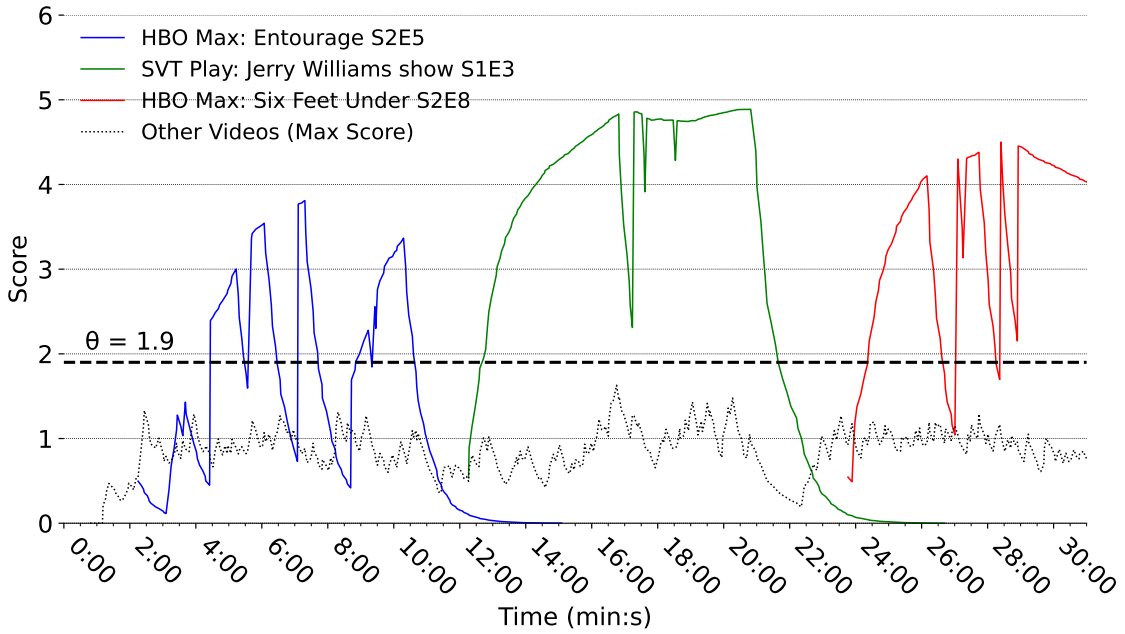


Figure 4.2: System scoring over time as target streams three videos with a VPN, demonstrating how it adapts over time. In this case, all streamed videos were identified.

the MacBook would sometimes be CAPTCHA prompted after logging in. Sessions that contained such a failure were discarded. Considering the overhead with our evaluation set-up, and that identifying even a single representation correctly against 550,000 others is noteworthy, we consider the size of the testing set comprising 667 randomly selected VPN streams and 270 randomly selected non-VPN streams to be adequate.

4.2 Results

The system's performance at any given time step t is determined by whether it outputs a correct prediction for representation i or returns -1 if confidence falls below the predetermined threshold θ. This non-conclusive approach necessitates an evaluation over time, particularly as the target content changes. In order to assess the system in a way that is interpretable, we account for the playback time in seconds $0 \leq t \leq 600$, and define the following three mutually exclusive outcomes for a single playback at time t:

- **Identified**(t) (abbrv. **Id.**): At least one correct prediction within time t with no incorrect predictions[2] inside the full 600 seconds watching session.
- **Misidentified**(t) (abbrv. **Mis.**): At least one incorrect prediction inside the watching session; constant for all t.
- **Unknown**(t) (abbrv. **Unk.**): Neither identified within t nor misidentified.

[2]Except during a 60 second grace period when the target switches to another video

Table 4.2: Closed-world VPN results after 10 minutes of playback.

Platform	Id.	Unk.	Mis.	Acc.	Mean Id. Time (min:s)
HBO Max	290	2	0	0.993	2:30
SVT Play	366	1	0	0.997	1:40
Prime Video	8	0	0	1.000	1:39
Total	664	3	0	0.995	1:56

Table 4.3: Closed-world Wi-Fi results after 10 minutes of playback.

Platform	Id.	Unk.	Mis.	Acc.	Mean Id. Time (min:s)
HBO Max	120	1	0	0.992	2:19
SVT Play	149	0	0	1.000	2:12
Total	269	1	0	0.996	2:15

Here, a correct prediction means that the predicted representation belongs to the video that is being streamed by the target, while an incorrect prediction belongs to some other video. When the target changes video, the score of the previous video will automatically decay. For this reason we allow a 60 second grace period, but only if the system's prediction is associated with the previous video played by the target. Additionally, since we classify a playback as misidentified if there is a single incorrect prediction inside the full 600 seconds, we can safely say that a video identified at some time $t = t_0$ will also be identified for all $t_0 < t \leq 600$.

4.2.1 Closed-world evaluation

We evaluate all streams against the system first in *closed-world* experiments, where each captured stream has a corresponding fingerprint in the k-d tree. This holds true for any randomly chosen HBO Max or SVT Play video, as their entire libraries are included in our fingerprint database. Consequently, the system should ideally identify all streamed videos in the testing set. Here, we measure accuracy (abbrv. Acc.) at the given time t as the proportion of identified streams at that time. Table 4.2 and Table 4.3 show the cumulative results at the full 10 minutes of playback time. The mean identification time indicates that most identifications happen earlier. Figure 4.3 shows the accuracy over time, with approximately 90% of streams being identified at the 3:30 mark for both VPN and Wi-Fi. Recall, the parameters used in this evaluation are the ones suggested in Table 3.3. There is no discernible discrepancy between the streaming devices; notably, the three videos streamed by the Samsung TV were all identified. Additionally, the Safari streams using HLS were also identified. While Wi-Fi streams generally exhibited longer identification times, we note that when using a VPN, HBO Max was considerably slower (beyond normal VPN overhead), and took longer to reach the steady state, delaying the identification. It is possible that HBO Max applies throttling on known VPN servers.

Investigating the unknown outcomes where the system failed to make decision, we

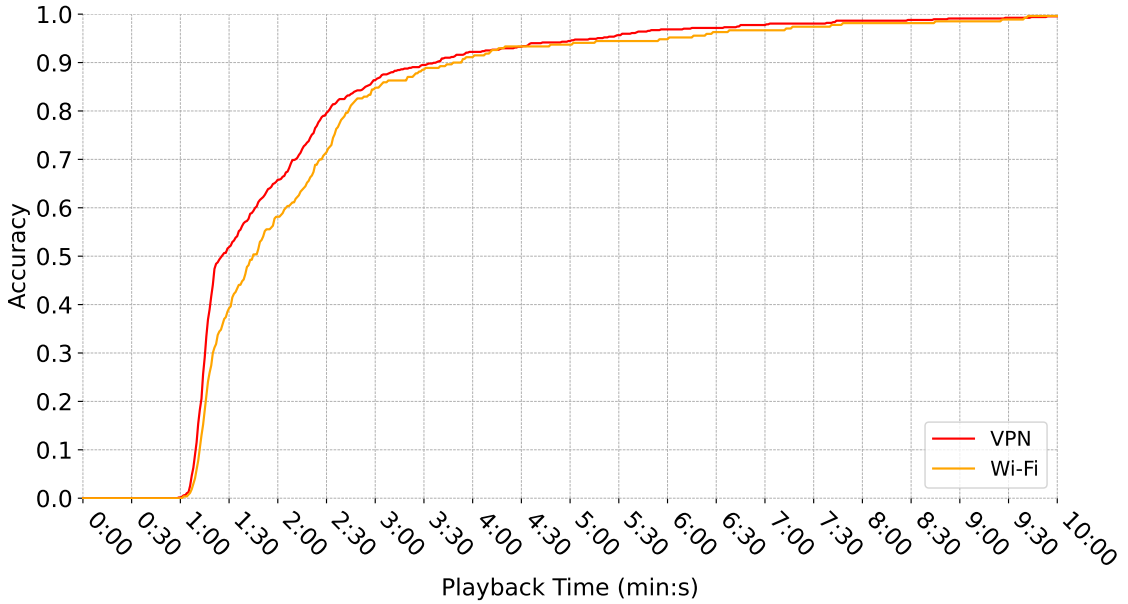


Figure 4.3: Closed-world cumulative accuracy for VPN and Wi-Fi over playback time.

note that in 3 out of 4 cases, the system considered these as candidates but that they did not reach the set threshold of $\theta = 1.9$. It is possible that they would have been identified with more time. The only remaining case was never considered by the system, and after investigation we found out that the video had been re-encoded and that the new fingerprint was not actually in our dataset. Such cases can be avoided by running the data collection program on a schedule (e.g. daily) to keep the fingerprint database up-to-date. The full results for each playback is available with our open-source implementation. An identification does not assess how consistently the system retains its initial correct prediction. The score will sometimes drop below the threshold after the identification has happened due to uncertainty, see Figure 4.2. For HBO Max, SVT Play and Prime Video, the system retains its prediction in 74.0%, 89.3% and 88.8% of the time steps, respectively, in the VPN scenario. In the Wi-Fi scenario, the corresponding figures are 79.8% for HBO Max and 88.1% for SVT Play.

4.2.2 Open-world evaluation

Unlike previous fingerprinting works, that could identify a streaming service by associating it with its known IP-addresses or by eavesdropping on TLS-handshakes to see the server name indication, we cannot know of the source of the traffic as it is tunneled through the VPN connection or when eavesdropping on the encrypted Wi-Fi link. As the traffic could be anything, including other streaming services not in our dataset, this necessitates evaluating our system in such an *open-world* setting. A benefit of our non-ML model and heuristic is that we do not have to do expensive model training or train e.g. an *unknown* class. We can simply re-build our k-d tree with a subset of videos removed to evaluate on completely unknown data to the

Table 4.4: Open-world Wi-Fi results after 10 minutes of playback.

Platform	**Id.**	**Unk.**	**Mis.**	**Unk.**$_R$	**Mis.**$_R$	**Acc.**	**Prec.**	**Rec.**
HBO Max	144	2	0	146	0	0.993	1.000	0.986
SVT Play	183	0	0	183	1	0.997	0.994	1.000
Prime Video	5	0	0	3	0	1.000	1.000	1.000
Total	332	2	0	332	1	0.995	0.998	0.994

Table 4.5: Open-world Wi-Fi results after 10 minutes of playback.

Platform	**Id.**	**Unk.**	**Mis.**	**Unk.**$_R$	**Mis.**$_R$	**Acc.**	**Prec.**	**Rec.**
HBO Max	60	1	0	60	0	0.991	1.000	0.983
SVT Play	74	0	0	75	0	1.000	1.000	1.000
Total	134	1	0	135	0	0.996	1.000	0.992

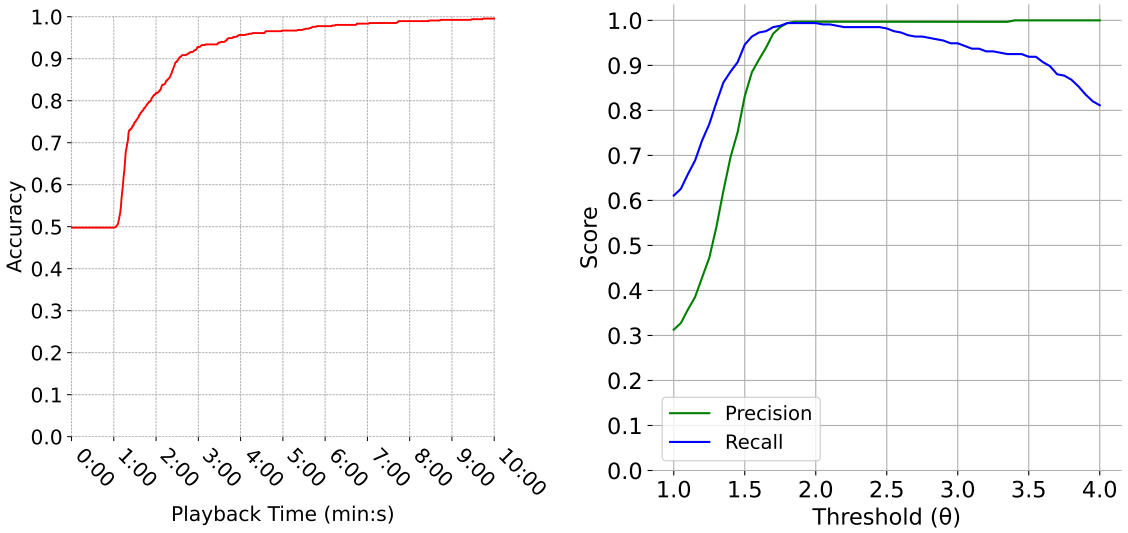
system. Therefore, we take randomly 50% of our testing set and remove all their representations from the dataset, creating a *removed* set of 333 VPN streams and 135 Wi-Fi streams. This strategy barely puts a dent in the dataset size, with the remaining dataset consisting of 44,713 videos and 545,310 fingerprints in the VPN scenario, a 0.7% difference. Furthermore, this tests the system on network data with a very similar profile.

In the open-world evaluation, we note that the streams from the *removed* set will have two outcomes, either unknown or misidentified, with the same definitions are before. We abbreviate these new outcomes with **Unk.**$_R$, the positive case, and **Mis.**$_R$, the negative case, respectively. Furthermore, we introduce *precision*, i.e. a measure of our system's capability of making qualitative predictions, punishing misidentifications, and *recall*, measuring the system's ability to capture known streams, at potential cost of misidentifying *removed* streams. We thus extend the metrics definition as follows:

$$\mathbf{Acc.}(t) = \frac{\sum_t \mathbf{Id.}(t) + \sum_t \mathbf{Unk._R}(t)}{\text{Total samples}}$$

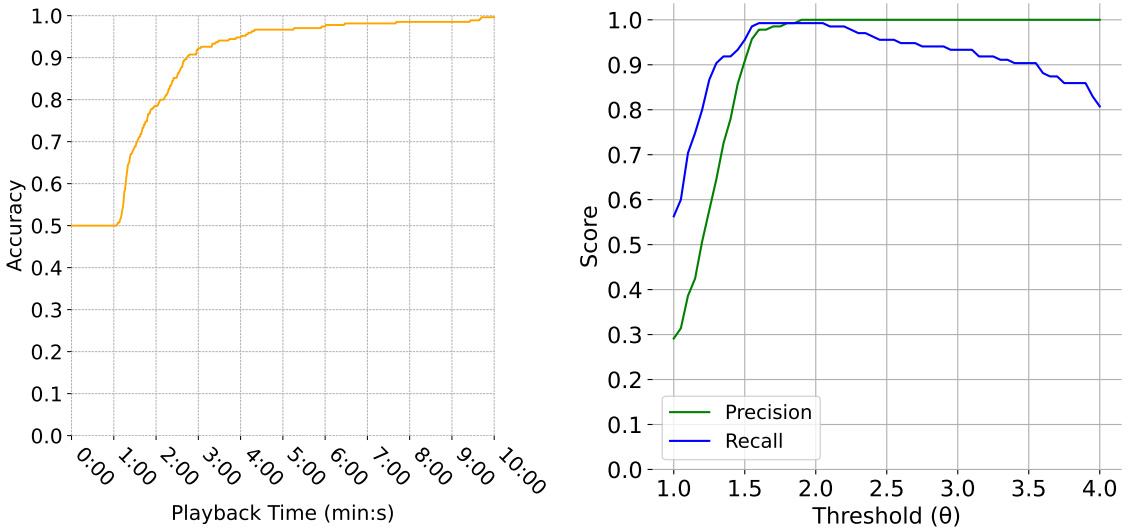
$$\mathbf{Prec.}(t) = \frac{\sum_t \mathbf{Id.}(t)}{\sum_t \mathbf{Id.}(t) + \sum_t \mathbf{Mis.}(t) + \sum_t \mathbf{Mis._R}(t)}$$

$$\mathbf{Rec.}(\mathbf{t}) = \frac{\sum_t \mathbf{Id.}(t)}{\sum_t \mathbf{Id.}(t) + \sum_t \mathbf{Mis.}(t) + \sum_t \mathbf{Unk.}(t)}$$



(a) Cumulative accuracy over playback time. (b) Prec. and rec. at 10 min for various θ.

Figure 4.4: Open-world VPN results.



(a) Cumulative accuracy over playback time. (b) Prec. and rec. at 10 min for various θ.

Figure 4.5: Open-world Wi-Fi results.

Table 4.4 and Table 4.5 show the results for the open-world evaluation, using the same parameters as before, demonstrating that the system generalizes well to this setting. Note that identification time and the retain figures do not deviate significantly from the closed-world evaluation, so they are excluded. Figure 4.4 and Figure 4.5 shows the accuracy over playback time, and the effect of varying confidence thresholds, for the VPN and Wi-Fi scenarios, respectively. Note accuracy now starting at approximately 0.5 due to the streams from the *removed* set being correctly classified as unknown from the start. We observe that the recall peaks around our selected $\theta = 1.9$ and that the system is already nearing 1.000 precision

at that point. Notably, we see a misidentification in the VPN evaluation, coming from the *removed* set. Investigating this shows that it came from the target streaming the SVT Play video *History of the World: Rebels of the Monarchy*, which was mistaken for another SVT Play Video, *Everything for the Monarchy: The Princess Who Became a Rebel* (titles translated). The misidentification is due to the latter containing identical footage for the majority of the streamed 10 minutes.

4.2.3 Comparison with previous method

We implemented the conclusive heuristic used by Björklund *et al.* [7], demonstrated to be 98.5% accurate for a dataset of 20,000 videos, having full network-level access. This method uses the k-d tree to retrieve a set of candidates that are subsequently compared with the query point using the Pearson correlation coefficient and declaring a match if it surpasses a predefined threshold. We evaluate the method using our open-world VPN streams. Table 4.6 demonstrates the difference in the quality of the input data in our attack model, reaching 0.368 recall in the best case, trying several thresholds among those suggested.

Table 4.6: Comparison with the method of Björklund *et al.* [7]. Open-world VPN results evaluating various Pearson thresholds.

Pearson Thr.	**Id.**	**Unk.**	**Mis.**	**Unk.**$_R$	**Mis.**$_R$	**Acc.**	**Prec.**	**Rec.**
1 - 1e-8	13	473	0	181	4	0.289	0.765	0.027
1 - 1e-7	136	371	0	160	25	0.441	0.845	0.268
1 - 1e-6	178	337	0	152	33	0.492	0.844	0.346
1 - 1e-5	187	330	0	150	35	0.502	0.842	0.362
1 - 1e-4	191	304	24	135	50	0.486	0.721	0.368
1 - 1e-3	107	38	359	19	166	0.188	0.169	0.212
1 - 1e-2	3	0	479	0	185	0.004	0.004	0.006

4.2.4 Summary

Our evaluation demonstrates the concerning aspect of the demonstrated privacy attack, and improvements over previous methods that relied on the network attacker model. Indeed, among a massive database of 45k videos and 550k fingerprints, 99.5% of streams are identified within 10 minutes with successful identification occurring on average within 2 minutes of starting our capturing program. These claims hold regardless if the targets are using a VPN to mask their traffic or if the attack has no network access but simply monitor the wireless traffic within reach. We further demonstrate that our recognition system do not misidentify streams that are not part of the dataset with precision and recall consistently above 0.99 regardless of the targeted streaming platform and device used by the user, be it a Windows or Mac laptop, or a smart TV.

5

Mitigation

This chapter explores how the attack outlined in this thesis can be mitigated. In § 5.1, we first examine a seemingly straightforward solution: manipulating network traffic to distort the bursty pattern caused by ABR streaming. Then, in § 5.2, we take a closer look at the underlying cause and propose changes that can be made to streaming clients to address the source of the leak.

5.1 Traffic Manipulation

Recent studies [12, 19, 24] have explored defensive measures against encrypted traffic analysis attacks, primarily focusing on website fingerprinting. These strategies include padding and delaying packets. However, for video streams, the padding would need to be substantial and applied in a non-trivial manner; e.g. constant padding could be negated by using a discrete derivative as outlined in this thesis.

The VPN service Mullvad recently introduced an optional feature called Defense against AI-guided Traffic Analysis (DAITA) in their beta client [17]. DAITA is built using the Maybenot framework, introduced by Pulls *et al.* [19]. It works by ensuring constant packet sizes, interspersing random background traffic, and distorting data patterns to make it difficult for observers to identify meaningful activity or specific website visits. To evaluate DAITA, we conducted an experiment by streaming five videos, each played twice for 10 minutes: once with a standard Mullvad VPN connection and once with DAITA activated. In each scenario, we measured the data transferred during playback, excluding the first 100 seconds of each capture to ensure steady state playback had been reached.

The results demonstrate that DAITA mitigates the attack outlined in this thesis. Our system did not identify the streamed videos with DAITA activated, whereas each video was successfully identified in the non-DAITA scenario. However, DAITA significantly increases data usage, as shown in Table 5.1. This increase has several implications: higher bandwidth usage can impact the quality of experience, potentially leading to lower quality streams; many ISPs impose data caps, which could limit usage; and perhaps more importantly, the energy costs associated with increased data usage must be considered. Nevertheless, we advocate for the development of such privacy-preserving features. DAITA, although in its early stages, shows promise and is likely to improve with further research. In its current state,

Table 5.1: Data transferred (down / up / total) over 500 seconds of steady state playback in the non-DAITA and DAITA scenario.

No.	Non-DAITA (MB)	DAITA (MB)	Increase (%)
1	240.5 / 1.8 / 242.2	403.9 / 53.9 / 457.8	67.9 / 3672.2 / 89.0
2	235.1 / 1.6 / 236.7	470.3 / 75.8 / 546.1	100.0 / 4637.5 / 130.7
3	252.6 / 1.6 / 254.2	553.0 / 69.2 / 622.2	118.9 / 4225.0 / 144.8
4	245.4 / 1.8 / 247.2	413.4 / 65.6 / 479.0	68.4 / 3544.4 / 93.8
5	232.9 / 1.6 / 234.5	453.5 / 56.8 / 510.3	94.7 / 3450.0 / 117.6
Mean	241.3 / 1.7 / 243.0	458.8 / 64.3 / 523.1	90.1 / 3682.4 / 116.8

DAITA is a viable option for mitigating the attack when privacy is paramount, but further research is required to thoroughly evaluate its efficacy.

5.2 Addressing the Leak Source

Traffic manipulation is a band-aid solution to the leak which stems from the design of ABR streaming clients. We could pad segments on the server side, but that would negate the advantages of using variable bitrate in the first place, such as reducing bandwidth costs and maintaining quality of experience which are top priorities for content providers [6]. ABR protocols are inherently client-driven, and we believe that is where the prevention strategy should lie. This section will discuss the cause of the leak and propose a number of conditions to prevent it.

5.2.1 Buffer management

In this thesis, we assume that one burst, excluding audio and other noise, corresponds to one video segment during the streaming steady state, meaning the buffer is full and network conditions are stable. This reflects a buffer strategy of a request being triggered when the buffer falls below a predefined threshold, typically a segment length, and seems to be the most common strategy from our observations. Consequently, this means that all network traces for a representation playback, during the steady state will have a single permutation, or mapping, to its fingerprint and lets us build a single k-d tree to find candidates for all such videos. A different strategy employed by some services is the use of shorter segments where the client requests multiple ones in quick succession. This can be beneficial for latency assuming a whole segment needs to be decoded to initiate playback, but has the drawback of increased load on servers because of the higher request frequency. Here, the requested segments in each "on period" depend on when the steady state is reached. This raises the question if the attacker can deal with a variable number of segments interleaved in a single burst.

5.2.2 HTTP

To address this, we assume a network eavesdropper that can observe the transport layer, albeit TLS-encrypted, conversation between the target and the streaming service, i.e. the model of many previous works [7, 9, 14, 23, 29]. The information that can be dissected from the multiple segments in the same on period depend on the HTTP version. It is known that HTTP/1.1 suffers from head-of-line blocking, and to overcome this, simultaneous requests for resources use multiple TCP connections. Consequently, if the client uses HTTP/1.1, the attacker can derive the individual segment sizes from the different connections. HTTP/2 offers multiplexing at the HTTP level, meaning resources can share a single TCP connection. HTTP/3 further improves on this, addressing some shortcomings of TCP itself and employs the UDP-based QUIC protocol instead, as mentioned in § 2.4. The use of HTTP/2 and HTTP/3 thus poses challenges to the attacker when multiple segments are requested, in that deriving their individual sizes from a single burst will be difficult.

In its current state, the attack outlined in this thesis would not work on a streaming service that employs a buffer management strategy of requesting a variable number of segments in each on period. However, as we discuss next, this is not a general countermeasure to the vulnerability and we do not recommend settling for such a solution.

5.2.3 Timing considerations

In this work, as in previous fingerprinting works, the variable bitrate encoded segment sizes have been the focus. Setting aside concerns about inefficiency, consider an encoding strategy where each representation is divided into segments of equal size. The observed bursts for all videos will thus be similar in size, effectively stopping known methods. However, this approach would only shift the focus of the attack to the time domain, since the segment durations would now vary significantly. Here, the attacker could use the time between bursts, and the fingerprint would comprise the segment durations. We note that this is possible because the buffer will, during the steady state, try to remain constant. This is generally true and not restricted to equal sized segments.

We believe that the buffer striving to stay at a constant size during the steady state is the *root cause* of the vulnerability and that there, with this assumption, is potential for a method that is also agnostic to the buffer strategy. For example, such a solution would incorporate both the segment sizes and the segment durations for a generalized fingerprint. This remains to be evaluated by future work, but as a proactive measure, we suggest the use of a *randomized* buffer. In this scenario, the buffer could be configured to maintain a capacity between a predefined lower bound, e.g. the previous max size of the buffer, and an upper bound. The buffer would then be allowed to deplete to a random level within this range before the next request phase is triggered which requests between 1 and N segments, ensuring that the number of segments requested does not cause the buffer to exceed its upper bound. This layer of randomness would challenge any fingerprinting system derived from constant features.

5.2.4 Summary

To prevent the attack, we propose three conditions that should be satisfied by a future client implementation: **(1)** a buffer strategy where multiple segments may be requested simultaneously, **(2)** use of HTTP/2 or higher, and **(3)** the use of a randomized buffer. For the evaluated attack models in this thesis, **(2)** is not required as the VPN or Wi-Fi attacker does not have access to such information. The efficacy of our proposed prevention strategy remains an area for future research.

6

Limitations and Future Directions

In this chapter, we discuss the limitations of our work and unevaluated scenarios that could warrant further investigation. While some of these limitations have been mentioned throughout the thesis, we will consolidate and elaborate on them in § 6.1-6.4. The discussed directions for future work are compiled and summarized in § 6.5.

6.1 Dataset

Recall, as outlined in § 3.1, data collection is fast once a client that retrieves metadata and manifests has been developed. However, developing such a client takes time. Although this thesis presents the largest dataset ever used for this purpose, we believe our system could scale to handle an even larger dataset. Due to time constraints, we needed to focus on other parts of the system. The time required to develop a client is highly dependent on the platform, with most of the work involving analyzing requests and responses for undocumented internal APIs. For example, developing a client for SVT Play, which has a straightforward design, took only a few hours. In contrast, HBO Max, with its more complex API, required a longer period to decipher. A brief investigation into other platforms suggests that e.g. Prime Video and YouTube should be relatively simple to handle. However, Netflix employs a security layer even on the client side, with requests and responses only decrypted by their application logic. This complicates things, but is essentially security by obscurity, and can be deciphered with some know-how (see netflix-mitm-proxy[1]), provided a paid account is used.

An interesting approach would be to leverage the capabilities of yt-dlp[2], which is designed to download streamable videos online to a local file on disk. By using its plugins for hundreds of (non-DRM) platforms, made possible by an open-source effort, it could obtain links to manifest files. Using this method, the dataset could potentially grow considerably large, with perhaps millions of fingerprints from multiple platforms. Additionally, synthetic data could be employed to push the dataset's limitations. For example, artificial fingerprints derived from real fingerprint datasets, although not identifiable from actual video streams, could still complicate the search for real streams. Furthermore, expanding the dataset may necessitate the use of multiple k-d trees or alternative solutions across multiple systems, making a distributed

[1] `https://github.com/nohajc/netflix-mitm-proxy`
[2] `https://github.com/yt-dlp/yt-dlp`

identification system an interesting area for exploration.

6.2 High-throughput Traffic

Our capture tool, BurstShark, processes output from TShark with minimal overhead. TShark, utilizing the libpcap library, is adept at analyzing network traffic, including headers and payloads. However, for the purposes of this traffic analysis attack, minimal information is required; essentially, just some IP details, packet size, and timing. TShark and libpcap are not designed for extremely high-throughput environments, such as ISP-level traffic. Therefore, although the thesis explores the potential for mass surveillance at the ISP level, the network tool developed for this thesis is unlikely to be effective in such scenarios. To handle high-throughput situations with a large number of concurrent streamers, more specialized software, for example kernel bypassing frameworks such as DPDK[3] or PF_RING[4], or even dedicated hardware, could be explored.

6.3 Network Activity and Conditions

In our VPN and Wi-Fi attack models, the target is assumed not to be actively engaging in other activities besides streaming. While we believe this to be a realistic scenario when a user is watching a video, it would be interesting to evaluate scenarios where the target simultaneously generates other data-heavy traffic, such as browsing the web. Furthermore, in our evaluation, the target has good bandwidth and generally receives the best quality available on its device, leaving lower quality representations (i.e., sub 540p) untested.

Unstable network conditions, as previously mentioned, could cause frequent switching of the representation. The method outlined in this thesis requires segments requested from the same representation continuously for a sufficient period of time (2-3 minutes on average, cf. Chapter 4). If the network is consistently unstable, identification would prove difficult or impossible, making this another possible direction for future research.

Privacy-preserving features that spoof and distort network activity, such as Mullvad's DAITA, warrant a more thorough evaluation than what is presented in this thesis. These features are primarily designed to defend against website fingerprinting, so it is possible that, given sufficient time, statistical analysis could still reveal the video content due to the number of requests. Additionally, evaluating the attack on other privacy-preserving measures than VPN, such as the Tor network[5], which also employs traffic manipulation techniques to mitigate traffic analysis, could be interesting.

[3] `https://github.com/DPDK/dpdk`
[4] `https://github.com/ntop/PF_RING`
[5] `https://www.torproject.org`

6.4 Generalized Attack and Defense

In the previous chapter, we discussed the impact of the buffer management strategy employed by the ABR client and proposed a potential generalized fingerprint that considers both segment sizes and durations. An initial approach could involve using time series analysis techniques, such as dynamic time warping, to match a fingerprint sequence against a video network trace. This generalized attack would necessitate a corresponding generalized mitigation strategy, which we also proposed in the previous chapter. However, this strategy needs to be evaluated. One possible method for evaluation is to set up a local video server and streaming client, such as a reference DASH client[6] with the proposed conditions implemented.

6.5 Summary

The following presents a summarized list of directions for future work as discussed above and throughout the thesis:

- Expand the dataset to explore scalability (cf. § 6.1)
- Investigate solutions for high-throughput traffic, i.e. viability of mass surveillance at ISP-level (cf. § 6.2)
- Explore identification for various network trace scenarios, e.g. multi-tasking targets or unstable network conditions (cf. § 6.3)
- Conduct thorough evaluation of privacy-preserving features such as Mullvad's DAITA or Tor (cf. § 5.1, § 6.3)
- Develop and assess generalized fingerprinting techniques and this thesis proposed mitigation strategy (cf. § 5.2, § 6.4)

[6] `https://github.com/Dash-Industry-Forum/dash.js`

7

Conclusion

7.1 Ethical Considerations

This thesis reveals a significant privacy issue with implications of covert surveillance, where individuals could be monitored without their knowledge through VPNs and Wi-Fi. A valid concern is whether publicizing this work does more harm than good. However, highlighting vulnerabilities in programs and protocols used by the public is important. It follows the cycle of uncovering weaknesses to strengthen our defenses against them, a fundamental aspect of advancing network security. By bringing vulnerabilities to light, the community can collaborate on solutions. Furthermore, the vulnerability is not a new, unknown (zero-day) threat. Rather, it is an extension of a known issue, now shown to be more severe. Considering that the leak has been known since 2016 [20], it is possible that actors with significant resources might already exploit it.

Additionally, this thesis serves an educational purpose, alerting individuals and organizations to potential privacy concerns inherent in current video streaming. If the vulnerability is known, even if not solved, users can at least make informed decisions considering privacy implications of streaming videos online. This work also discusses mitigation strategies, providing a positive contribution by suggesting ways to address the vulnerabilities identified. It is important to navigate the line between the need to disclose vulnerabilities for the sake of improving security and the potential misuse of this information. However, by following ethical disclosure protocols and focusing on public education, this thesis contributes to the ongoing effort to enhance digital privacy and network security.

7.2 Concluding Remarks

We have in this thesis presented a concerning narrative about the video streaming landscape, demonstrating that the DASH leak is more extensive than previously documented. Our system shows remarkable accuracy and precision at scale, and could be used to identify if a streamer is watching specific videos deemed undesirable or controversial, even when they have taken the privacy-preserving measure of using a VPN. Being the first such system capable of scaling to datasets of several thousands of videos with ca. 10 representations per video and working regardless if a VPN is used or if the attacker lacks network access, we believe our thorough evaluation

demonstrates the robustness of our recognition method.

A strength of our work is the experimental setup and rigorous evaluation, supported by our open-source toolkit and dataset. These resources promote transparency and fully reproducible experiments, enabling future researchers to use our public dataset and captured data without needing to establish the entire network themselves. Additionally, the background and detailed discussion in the mitigation chapter provide a good foundation on adaptive bitrate streaming, making it more accessible than many previous works, which often overlook this aspect due to different priorities or a lack of understanding. Consequently, this thesis serves as a solid basis for future research in this domain, for which we propose several directions.

One of the challenging aspects of this thesis was working with live systems that we did not control. This introduced difficulties, as different target platforms required varied strategies for data collection and automated playback for evaluation. Many sessions were discarded due to failures from unforeseen circumstances, such as simple pop-ups that a real user could easily dismiss but would crash the script. Despite the challenges of having less control over the test environment, this aspect is also a strength of our work, as it reflects real-world scenarios. Setting up a local environment with a streaming server might raise questions about its applicability in such scenarios and also not attract enough attention to the leak.

The privacy implications of this thesis are very substantial, extending beyond mere privacy invasion to potential surveillance, censorship or discriminatory treatment. We hope our work raises public awareness about the vulnerabilities in video streaming, encouraging informed consumer choices. At last, we would like to call on streaming developers to take our proposed mitigation strategy into account and develop solutions promptly to address the leak.

Bibliography

[1] ISO/IEC 23000-19:2024. *Multimedia application format (MPEG-A) - Part 19: Common media application format (CMAF) for segmented media*. Standard. International Organization for Standardization, Feb. 2024, pp. 1–142. URL: https://www.iso.org/standard/85623.html.

[2] ISO/IEC 23001-7:2023. *MPEG systems technologies - Part 7: Common encryption in ISO base media file format files*. Standard. International Organization for Standardization, Aug. 2023, pp. 1–42. URL: https://www.iso.org/standard/84637.html.

[3] ISO/IEC 23009-1:2022. *Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats*. Standard. International Organization for Standardization, Aug. 2022, pp. 1–320. URL: https://www.iso.org/standard/83314.html.

[4] Waleed Afandi et al. "Fingerprinting technique for youtube videos identification in network traffic". In: *IEEE Access* 10 (2022), pp. 76731–76741.

[5] Sangwook Bae et al. "Watching the Watchers: Practical Video Identification Attack in LTE Networks". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 1307–1324. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/bae.

[6] Bitmovin. *The 7th Annual Video Developer Report*. Tech. rep. Bitmovin, Jan. 2024, pp. 1–34. URL: https://bitmovin.com/downloads/assets/bitmovin-7th-video-developer-report-2023-2024.pdf.

[7] Martin Björklund et al. "I see what you're watching on your streaming service: Fast identification of dash encrypted network traces". In: *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. IEEE. 2023, pp. 1116–1122.

[8] Meijie Du et al. "Long-Short Terms Frequency: A Method for Encrypted Video Streaming Identification". In: *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE. 2023, pp. 1581–1588.

[9] Ran Dubin et al. "I know what you saw last minute-encrypted http adaptive video streaming title classification". In: *IEEE Transactions on Information Forensics and Security* 12.12 (2017), pp. 3039–3049.

[10] Romaric Duvignau. "Metainformation Extraction from Encrypted Streaming Video Packet Traces". In: *Proc. of the International Conference on Electri-*

cal, Computer, Communications and Mechatronics Engineering (ICECCME). IEEE. 2022, forthcoming.

[11] DASH Industry Forum. *DASH-IF implementation guidelines: restricted timing model*. Tech. rep. DASH Industry Forum, June 2020, pp. 1–51. URL: https://dashif-documents.azurewebsites.net/Guidelines-TimingModel/master/Guidelines-TimingModel.pdf.

[12] Jiajun Gong et al. "WFDefProxy: Real World Implementation and Evaluation of Website Fingerprinting Defenses". In: *IEEE Transactions on Information Forensics and Security* 19 (2024), pp. 1357–1371.

[13] Google. *Google Transparency Report - HTTPS encryption on the web*. 2024. URL: https://transparencyreport.google.com/https/overview.

[14] Jiaxi Gu et al. "Traffic-based side-channel attack in video streaming". In: *IEEE/ACM Transactions on Networking* 27.3 (2019), pp. 972–985.

[15] Egl Juodyt. *NordLynx protocol the solution for a fast and secure VPN connection*. Jan. 2022. URL: https://nordvpn.com/blog/nordlynx-protocol-wireguard.

[16] Muhammad US Khan et al. "Scnn-attack: A side-channel attack to identify youtube videos in a vpn and non-vpn network traffic". In: *Electronics* 11.3 (2022), p. 350.

[17] Mullvad. *Introducing Defense against AI-guided Traffic Analysis (DAITA)*. May 2024. URL: https://mullvad.net/en/blog/introducing-defense-against-ai-guided-traffic-analysis-daita.

[18] Roger Pantos and William May. *HTTP Live Streaming*. RFC 8216. RFC Editor, Aug. 2017, pp. 1–60. URL: https://www.rfc-editor.org/rfc/rfc1654.txt.

[19] Tobias Pulls and Ethan Witwer. "Maybenot: A Framework for Traffic Analysis Defenses". In: *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*. WPES '23. Association for Computing Machinery, 2023, pp. 75–89. ISBN: 9798400702358.

[20] Andrew Reed and Benjamin Klimkowski. "Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections". In: *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2016, pp. 1107–1112.

[21] Andrew Reed and Michael Kranch. "Identifying https-protected netflix videos in real-time". In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 2017, pp. 361–368.

[22] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. RFC Editor, Aug. 2018, pp. 1–160. URL: https://www.rfc-editor.org/info/rfc8446.

[23] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. "Beauty and the burst: Remote identification of encrypted video streams". In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1357–1374.

[24] Jean-Pierre Smith et al. "QCSD: A QUIC Client-Side Website-Fingerprinting Defence Framework". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 771–789. ISBN: 978-1-939133-31-1.

[25] Iraj Sodagar. “The mpeg-dash standard for multimedia streaming over the internet”. In: *IEEE multimedia* 18.4 (2011), pp. 62–67.

[26] Weitao Tang et al. “Shrink: Identification of Encrypted Video Traffic Based on QUIC”. In: *2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE. 2023, pp. 140–149.

[27] Aliza Vigderman and Gabe Turner. *The Best VPN Services of 2024*. Apr. 2024. URL: `https://www.security.org/vpn/best`.

[28] Mingkai Wang et al. “Noise-Resistant Video Channel Identification”. In: *Security and Communication Networks* 2022 (2022).

[29] Hua Wu et al. “Identification of encrypted video streaming based on differential fingerprints”. In: *IEEE Conference on Computer Commun. Workshops (INFOCOM WKSHPS)*. 2020, pp. 74–79.

[30] Luming Yang et al. “Markov Probability Fingerprints: A Method for Identifying Encrypted Video Traffic”. In: *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE. 2020, pp. 283–290.

[31] Xiyuan Zhang et al. “Traffic spills the beans: A robust video identification attack against YouTube”. In: *Computers & Security* 137 (2024), p. 103623.

A

Test Environment

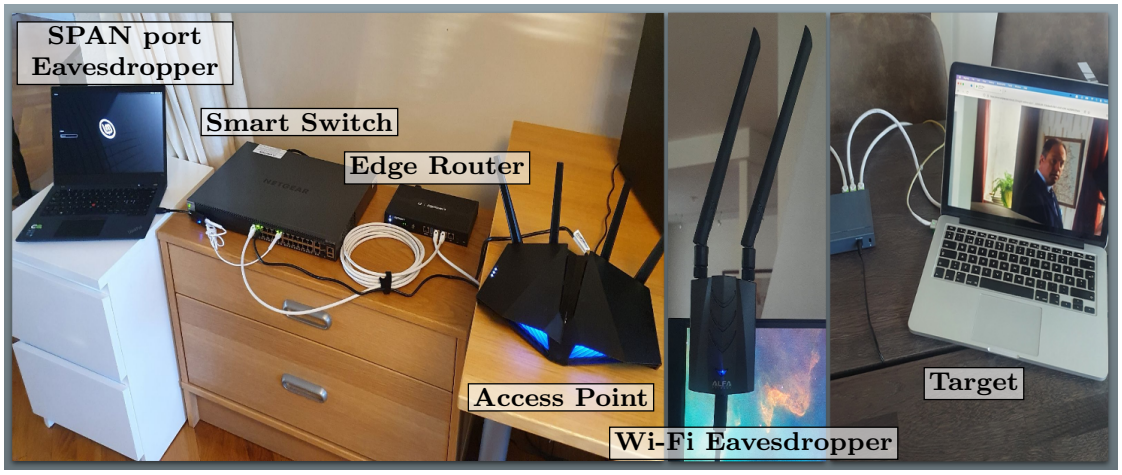


Figure A.1: Deployed experimental setting.